ELSEVIER

Science of Computer Programming 39 (2001) 215–247

Science of Computer Programming

www.elsevier.nl/locate/scico

Analysis of three hybrid systems in timed μCRL

Jan Friso Groote[a,b,*], Jos van Wamel[a]

[a] *CWI, P.O. Box 94079, 1090 GB Amsterdam, Netherlands*

[b] *Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, Netherlands*

Received 1 June 1999; received in revised form 5 January 2000; accepted 9 February 2000

Abstract

We study three simple hybrid control systems in timed μCRL [6]. A temperature regulation system, a bottle filling system and a railway gate control system are specified component-wise and expanded to linear process equations. Some basic properties of the systems are analysed and a few correctness requirements are proven to be satisfied. Although not designed for this purpose, timed μCRL seems to allow detailed analysis and verification of hybrid systems. The operators for parallelism and encapsulation are handled using some basic results from [10]. It turns out that the expansion and encapsulation of a parallel composition of processes generally leads to a considerable number of potential *time deadlocks*, which generally turn out to be harmless. Also inherent to parallelism are the multiple time dependencies between the summands of the separate components. As a consequence, expansions tend to lead to large numbers of terms. Various techniques, such as the use of invariants [5], have to be employed to master these complications.

1. Introduction

In order to deal with systems that use explicit time references in a process algebraic way, serious efforts have been made in the past. We recall, for instance, the formalisms defined in [3] (real-time process algebra), and [4] (discrete-time process algebra). As relevant formalisms with time from other lineages we mention [1,14–17].

A recent development is *timed* μCRL [6], which forms an extension of the language μCRL [7]. The reason why timed μCRL was developed, while already two related formalisms existed, was that timed μCRL appears to have certain advantages over the existing formalisms.

For instance, μCRL provides a variable binding construct, conditionals, and all facilities for reasoning with processes parameterised with data terms [8]. Therefore, not

* Corresponding author. Tel.: +31-20-5924232; fax: +31-20-5924199.
E-mail addresses: jfg@cwi.nl (J.F. Groote), jos@cwi.nl (J. van Wamel).

0167-6423/01/$ - see front matter
PII: S0167-6423(00)00010-1

much additional theory was needed and time could be incorporated in μCRL as an abstract data type. Basically, one new operator had to be added: the binary *at* operator ($@$). The expression $x@t$ stands for process x, where the initial actions happen at time t. The expressiveness of timed μCRL seems to be at least as big as that of comparable formalisms.

Many verifications have been made in μCRL, so that much experience and techniques are already available. Much of this is expected to generalise easily to the timed variant. One reason to believe that this will be the case is that timed μCRL was designed in such a way that a specification without references to time has the same intuitive meaning as a similar specification in the untimed case. Actually, we experienced that the calculations in this paper have the same 'look and feel' as many studies in untimed μCRL. The underlying principles, however, are much more intricate, and require a deeper understanding of the formalism.

Therefore, the first serious exercises in timed μCRL appeared separately in a recent paper [10]. In that paper various basic results were derived, such as theorems for *basic forms*, the expansion of terms with operators for parallelism, elimination of parallelism, and commutativity of the merge and communication merge (the operators $\|$ and $|$). In this paper, associativity of both these operators is included in the form of axioms. The results in [10] are directly applicable to the *linear process expressions* we use in this paper. We included a brief summary of useful data on timed μCRL, mainly from [10], in Appendices A and B.

This paper contains the first case studies in timed μCRL, and, considering the popularity and relevance of the subject, we choose to study three *hybrid control systems* of quite different kinds.

Hybrid control systems are classified as systems that combine the control of discrete event sequences with the control of continuous processes. Discrete events are, for instance, switches, incoming and outgoing message sequences, all kinds of human interaction with a system, etc. Continuous control usually concerns the control of processes governed by physical laws through differential equations, describing continuous relations between physical parameters such as time, place, temperature, voltage, pressure, electro magnetical field strength, etc. In practice, hybrid system theory can be said to comprise the study of the discrete control of continuous processes.

The first example we provide is about a temperature regulation system. It consists of a single process, so no parallelism is involved yet. This example, borrowed from [12], simply serves as a 'warming up'. In contrast with the analysis in [12], where modal formulas on the system behaviour are checked, we are able to analyse the system exactly.

The second example concerns a bottle filling system, consisting of two components: a conveyor belt with bottles and a container with liquid. The parallel composition is expanded to a single linear process, and the behaviour of the total bottle filling system, including the performance, is analysed in detail.

In the third example we study a railroad gate control system from [2]. Three processes are involved: A process which describes the passing trains, a controller, and

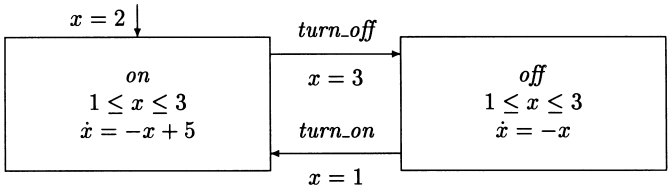

Fig. 1. The thermostat automaton.

gates. Again various correctness requirements are proven to be satisfied, for instance, that a train can never pass when the gates are open. In essence, we apply the same techniques as in the preceding example, although the analysis is considerably more involved.

For linearisation in the latter two examples we simply have to apply the Expansion Theorem from [10], and for the application of *encapsulation* to linear processes we have a general result in Appendix B. It turns out that encapsulation generally produces a number of *time deadlocks*, which are often redundant, but not always; they may reveal relevant system errors. In our examples, various techniques have to be employed to get rid of them, the most effective of which are invariants [5].

Our railroad example lies in the line of research described in [11], where a *generalised railroad crossing* is defined, specified and verified in the formalism of *timed automata*. As far as a comparison is reasonable, we do not think that we may claim a substantially easier or shorter way of analysing processes like the railroad gate controller. We do think, however, that in principle, our approach is more mechanical, which is due to the algebraic character of the analysis. We therefore expect that $\mu\mathrm{CRL}_t$ will become more significant for the analysis of timed systems as soon as more tools become available.

2. A thermostat

A small standard example of a hybrid system is given in [12]. It models a simple thermostat that keeps the temperature between 1° and 3°. In Fig. 1 the automaton is depicted.

The thermostat behaves as follows. Initially, the temperature is 2° and the heating is on. The temperature x in the room changes according to the differential equation $\dot{x} = -x + 5$. So it will go up. When the temperature has reached 3°, the *turn_off* action will take place, switching the heating off. The temperature will now drop according to the differential equation $\dot{x} = -x$. If the temperature has reached 1° the heater will turn on again, which is represented by the *turn_on* action.

In [12] it is shown how the HyTech tool can be used to check modal formulas. The authors show, for instance, that their tool can prove a formula stating that the heating is on for less than 2/3 of the total time. Using timed μCRL, the exact ratio $\ln 2/\ln 6$ (≈ 0.387) easily follows from the system equation.

The behaviour of the thermostat is specified below in timed μCRL. The system has two states; *on* and *off*, described by the data type *OnOff*. The variable t describes the time at which the system enters one of these states, and x describes the temperature at that instant. If the system is in state *on*, we want to have a *turn_off* action at some time u as soon as the temperature equals 3°, modelled by $f(u) = 3$, where the function f describes the variation of the temperature in time.

It is typical for the description of the thermostat that f is only described by a property, namely that the derivative of f equals $-f + 5$. Therefore, we use the sum operator to express that we are interested in any function f that satisfies this differential equation and the side condition $f(t) = x$.

In order to avoid confusion between bound and free variables, we assume a differential operator on functions, written as an accent, and use lambda notation. So, $f' = \lambda t. -f(t) + 5$ expresses what is written in Fig. 1 as $\dot{x} = -x + 5$.

Similarly, the system should do a *turn_on* action when $s = off$ and the temperature has dropped to 1°, where the temperature fall is described by the differential equation $\dot{x} = -x$. Note that the invariant condition $1 \leqslant x \leqslant 3$ is not described in process *Th* below, because it is satisfied implicitly.

$$
\begin{aligned}
\textbf{proc}\quad & Th(t{:}\textbf{Time}, x{:}\mathbb{R}, s{:}OnOff) = \\
& \overline{\textstyle\sum}_{f:Func,u:\textbf{Time}}\, turn\_off \lessdot u\; Th(u, 3, off) \\
& \quad \lhd s = on \wedge f' = \lambda t. -f(t) + 5 \wedge f(t) = x \wedge f(u) = 3 \rhd \delta \lessdot \mathbf{0} + \\
& \overline{\textstyle\sum}_{f:Func,u:\textbf{Time}}\, turn\_on \lessdot u\; Th(u, 1, on) \\
& \quad \lhd s = off \wedge f' = \lambda t. -f(t) \wedge f(t) = x \wedge f(u) = 1 \rhd \delta \lessdot \mathbf{0}
\end{aligned}
$$

where $\overline{\sum}_{f:Func,u:\textbf{Time}}$ abbreviates $\sum_{f:Func} \sum_{u:\textbf{Time}}$.

We want to understand this description better, and therefore we simplify it by applying the Sum Elimination Theorem (Appendix A.1). By standard mathematical analysis we know that there is a unique function f satisfying $f' = \lambda t. -f(t) + 5$ and $f(t) = x$. Without going into details on finding the solution, we state that f is given by $f(u) = (x - 5)\mathrm{e}^{t-u} + 5$. Similarly, the function f satisfying $f' = \lambda t. -f(t)$ and $f(t) = x$ is $f(u) = x\mathrm{e}^{t-u}$. Using the Sum Elimination Theorem we may simplify the previous equation to:

$$
\begin{aligned}
\textbf{proc}\quad & Th(t{:}\textbf{Time}, x{:}\mathbb{R}, s{:}OnOff) = \\
& \textstyle\sum_{u:\textbf{Time}}\, turn\_off \lessdot u\; Th(u, 3, off) \\
& \quad \lhd s = on \wedge (x - 5)\mathrm{e}^{t-u} = -2 \rhd \delta \lessdot \mathbf{0} + \\
& \textstyle\sum_{u:\textbf{Time}}\, turn\_on \lessdot u\; Th(u, 1, on) \\
& \quad \lhd s = off \wedge x\mathrm{e}^{t-u} = 1 \rhd \delta \lessdot \mathbf{0}
\end{aligned}
$$

For the first summand of the previous equation, we can derive that $u = t + \ln((5 - x)/2)$. For the second summand it follows that $u = t + \ln x$. Applying the Sum Elimination Theorem again, we obtain

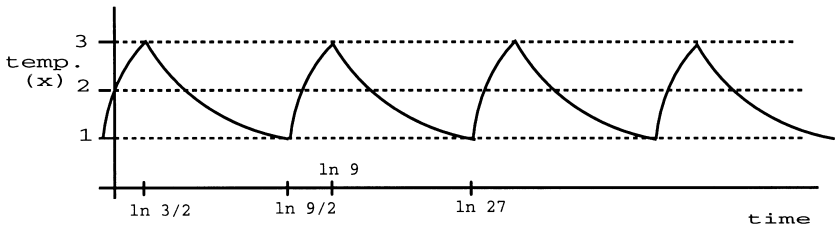

Fig. 2. Temperature versus time.

$$\textbf{proc}\quad Th(t{:}\textbf{Time}, x{:}\mathbb{R}, s{:}OnOff) = turn\_off\lhd(t+\ln(\tfrac{5-x}{2}))Th(t+\ln(\tfrac{5-x}{2}),3,off)\lhd s=on\ \rhd\delta\lhd\mathbf{0} + turn\_on\lhd(t+\ln x)\,Th(t+\ln x,1,on)\lhd s=off\rhd\delta\lhd\mathbf{0}$$

Process $Th(\mathbf{0},2,on)$ describes the thermostat starting at time $\mathbf{0}$, at temperature 2, with the heating on.

Now let

$$\textbf{proc}\quad Init = turn\_off\lhd\ln\tfrac{3}{2}\,Th'(\ln\tfrac{3}{2})$$
$$Th'(t{:}\textbf{Time}) = turn\_on\lhd(t+\ln 3)\,turn\_off\lhd(t+\ln 6)\,Th'(t+\ln 6)$$

Using the Recursive Specification Principle from process algebra (Appendix A.4) it easily follows that $Th(\mathbf{0},2,on)$=$Init$. So our final specification of the thermostat automaton exactly describes the moments where it switches between the states *on* and *off*. From the specification it is obvious that, eventually, the heater is on for a fraction $\ln 2/\ln 6$ of the time. Fig. 2 shows the relation between the temperature and the time.

3. A bottle filling system

3.1. Specification

We describe a bottle filling system with a buffer container as depicted in Fig. 3. Ten litre bottles are on a conveyor belt, above which there is an m litre container with some kind of liquid. When a bottle is under the container a tap is opened, and the liquid pours from the container into the bottles at a rate of 3 l/s. If a bottle is full the tap is closed and the conveyor belt starts moving. The next bottle takes 1 s to arrive. The container is filled at a constant rate of r $(2\leqslant r<3)$ litres per second.

The major question to be answered about this system, is under which conditions the container will overflow or get empty, when the system starts with a half full container at some time t.

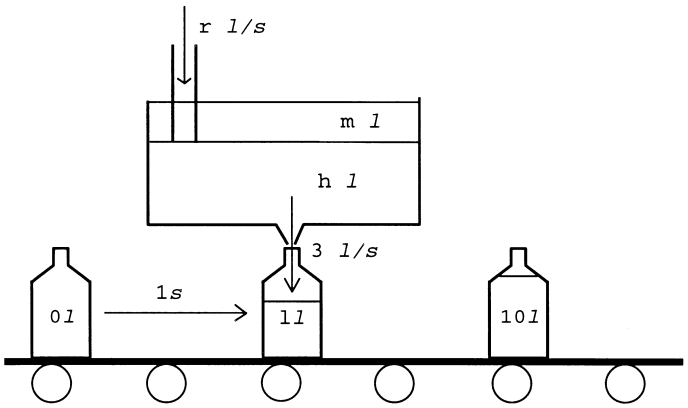

Fig. 3. The bottle filling system.

For a description in timed μCRL we have chosen for two parallel processes. One, described by a recursive equation defining the process CB, describes the conveyor belt with the bottles. The other, described by Con, describes the behaviour of the container.

We first describe the behaviour of CB in the various states of sort $CBState \stackrel{\text{def}}{=} \{move, nfill, sfill\}$ in detail:

1. $CB(t_b, l, move)$ denotes the state of the conveyor belt where one bottle has just been filled, and the next bottle starts moving towards the tap. At time t_b+1 it has reached the tap, and it indicates by an action $start_b$ that the (normal) filling starts. After this it behaves as $CB(t_b + 1, 0, nfill)$, i.e., the conveyor belt at time $t_b + 1$ in state $nfill$. The bottle under the tap is empty ($l = 0$).
2. The term $CB(t_b, l, nfill)$ represents the process where a bottle is being filled from time t_b off at 3 l/s. If the bottle is full, which takes place at a time t for which $3(t - t_b) = 10$, a $stop_b$ action indicates that the filling should stop. It could also be that the container becomes empty before the bottle is full, and this is indicated by an $empty_b$ action. From this moment the bottle is being filled at only r litres per second. Note that in state $nfill$ the CB process contains some non-determinism: At time $t_b + \frac{10}{3}$ the CB process may generate a $stop_b$ action, or it may receive an $empty_b$ signal from the container.
3. $CB(t_b, l, sfill)$ describes the conveyor belt with a bottle that is (slowly) being filled at r litres per second, where t_b is the moment when the container became empty, and l the liquid level in the bottle at that moment. Clearly, a $stop_b$ action must take place when the bottle is full. The moment t when this should happen is described by $l + r(t - t_b) = 10$.

proc $CB(t_b{:}\mathbf{Time}, l{:}\mathbb{R}, s_b{:}CBState) =$

(CB1) $start_b{\cdot}(t_b + 1)\, CB(t_b + 1, 0, nfill)$
$\quad \lhd s_b = move \rhd \delta{\cdot}\mathbf{0} +$

(CB2) $stop_b{}^{\varsigma}(t_b + \frac{10}{3})\, CB(t_b + \frac{10}{3}, 0, move)$

$\quad \lhd s_b = nfill \rhd \delta^{\varsigma}\mathbf{0} +$

(CB3) $\sum_{t:\mathbf{Time}} empty_b{}^{\varsigma}t\, CB(t, 3(t - t_b), sfill)$

$\quad \lhd s_b = nfill \wedge 3(t - t_b) \leqslant 10 \rhd \delta^{\varsigma}\mathbf{0} +$

(CB4) $stop_b{}^{\varsigma}(t_b + \frac{10-l}{r})\, CB(t_b + \frac{10-l}{r}, 0, move)$

$\quad \lhd s_b = sfill \rhd \delta^{\varsigma}\mathbf{0}$

We now describe the behaviour of the container in the various container states specified by sort $CState \stackrel{\text{def}}{=} \{inc, dec, dry\}$:

1. The process $Con(t_c, h, inc)$ represents the state of the container with the tap closed, from time t_c onwards. Parameter h denotes the container contents at time t_c. Clearly, at time u satisfying $h + r(u - t_c) = m$, where m is the capacity of the container, the container starts to run over. (In the specification below, m is treated as a constant.) As this is a 'dramatic' action, the behaviour of the system is not further described, but characterised with a time deadlock. In correct operation, of course, the tap will have to be opened in time by a $start_c$ action.
2. $Con(t_c, h, dec)$ describes the non-empty container with the tap open. The parameter h again represents the contents of the container at time t_c. The container may either become empty at time u, where u satisfies $h + r(u - t_c) - 3(u - t_c) = 0$, or stop filling a bottle before that moment.
3. $Con(t_c, h, dry)$ describes the container when it is empty while the tap is open. The liquid that pours in immediately pours out again, until it is indicated that the tap should close. Closing the tap brings the container back to state $Con(t_c, 0, inc)$.

We introduce two constants:

- $\tau_f \stackrel{\text{def}}{=} (m - h)/r$, which is the number of seconds before a container with a closed filling tap is full, and
- $\tau_e \stackrel{\text{def}}{=} h/(3 - r)$, which is the number of seconds before a container with an open filling tap is empty.

proc $Con(t_c{:}\mathbf{Time}, h{:}\mathbb{R}, s_c{:}CState) =$

(C1) $\sum_{u:\mathbf{Time}} start_c{}^{\varsigma}u\, Con(u, h + r(u - t_c), dec)$

$\quad \lhd s_c = inc \wedge h + r(u - t_c) < m \rhd \delta^{\varsigma}\mathbf{0} +$

(C2) $overflow^{\varsigma}(t_c + \tau_f)\, \delta^{\varsigma}(t_c + \tau_f)$

$\quad \lhd s_c = inc \rhd \delta^{\varsigma}\mathbf{0} +$

(C3) $\sum_{u:\mathbf{Time}} stop_c{}^{\varsigma}u\, Con(u, h - (3 - r)(u - t_c), inc)$

$\quad \lhd s_c = dec \wedge u < \tau_e + t_c \rhd \delta^{\varsigma}\mathbf{0} +$

(C4) $empty_c(t_c + \tau_e)\, Con(t_c + \tau_e, 0, dry)$
$\quad \lhd s_c = dec \rhd \delta \lhd \mathbf{0} +$

(C5) $\sum_{u:\mathbf{Time}} stop_c(u)\, Con(u, 0, inc)$
$\quad \lhd s_c = dry \rhd \delta \lhd \mathbf{0}$

The total system can be described by the parallel composition of the conveyor belt and container processes, where the synchronisation between these components is enforced by the ∂_H-operator.

proc $BFS(t_b{:}\mathbf{Time}, l{:}\mathbb{R}, s_b{:}CBState, t_c{:}\mathbf{Time}, h{:}\mathbb{R}, s_c{:}CState)$
$\quad = \partial_H(CB(t_b, l, s_b) \,\|\, Con(t_c, h, s_c))$

The variables t_b and t_c refer to the local time in *CB* and *Con*, respectively, $H \stackrel{\text{def}}{=} \{start_b, start_c, stop_b, stop_c, empty_b, empty_c\}$, and communications are defined by

comm $start_b \mid start_c = start$
$stop_b \mid stop_c = stop$
$empty_b \mid empty_c = empty$

3.2. A linearised variant

In Appendix B general equations are provided for the expansion of the parallel composition of two processes in linear format to another linear equation. In the same appendix it is shown how encapsulation may be applied to the resulting process. For the purpose of combined linearisation and encapsulation it is convenient to consider each pair of subterms from *CB* and *Con* separately.

When the processes *CB* and *Con* are put in parallel, each pair of summands CBi, Cj generates a transformation of the state variables s_b and s_c, e.g., CB1 and C1 may communicate and transform s_b, s_c from $move, inc$ to $nfill, dec$, respectively. In general, also additional constraints should be satisfied in order for the transition to take place. In our analysis this kind of state information, in conjunction with an invariant turns out to be very useful.

For proving an invariant of $BFS(t_b, l, s_b, t_c, h, s_c)$ correct it suffices to only consider the non-δ summands. This is because the δ-summands do not lead to new states. It turns out that if we start from states that satisfy $s_b{=}move \wedge s_c{=}inc$ the system can *possibly* only reach states that satisfy $s_b{=}move \wedge s_c{=}inc$, $s_b{=}nfill \wedge s_c{=}dec$ or $s_b{=}sfill \wedge s_c{=}dry$, which corresponds to our intuition.

As invariant we may take the disjunction of the above 3 states. Analysis learns that this invariant is vital for the cancellation of many (δ-)summands. In this bottle filling example, a full expansion would yield 46 terms, whereas an expansion using the invariant leads to only 18 terms!

Given that the invariant holds, process $BFS(t_b, l, s_b, t_c, h, s_c)$ may be characterised by the following summands:

CB1,C1.

c-summand:

(1)* $start^{<}(t_b + 1)\, BFS(t_b + 1, 0, nfill, t_b + 1, h + r(t_b - t_c + 1), dec)$
$\lhd s_b = move \wedge s_c = inc \wedge t_b - t_c < \tau_f - 1 \rhd \delta^{<}\mathbf{0}$

δ_{CB}-summand:

(2) $\delta^{<}(t_b + 1) \lhd s_b = move \wedge s_c = inc \wedge t_b - t_c < \tau_f - 1 \rhd \delta^{<}\mathbf{0}$ ($\subseteq$ term 1)

δ_{Con}-summand:

(3) $\sum_{u:\mathbf{Time}} \delta^{<}u \lhd s_b = move \wedge s_c = inc \wedge u$
$\leqslant t_b + 1 \wedge u < t_c + \tau_f \rhd \delta^{<}\mathbf{0}$ ($\subseteq 1 + 4$)

CB1,C2.

autonomous *Con*-summand:

(4)* $overflow^{<}(t_c + \tau_f)\, \partial_H(CB(t_b, l, s_b) \,\|\, (t_c + \tau_f) \gg \delta^{<}(t_c + \tau_f))$
$\lhd s_b = move \wedge s_c = inc \wedge t_c - t_b \leqslant 1 - \tau_f \rhd \delta^{<}\mathbf{0}$

δ_{CB}-summand:

(5) $\delta^{<}(t_b + 1) \lhd s_b = move \wedge s_c = inc \wedge t_b - t_c \leqslant \tau_f - 1 \rhd \delta^{<}\mathbf{0}$ ($\subseteq 1 + 4$)

CB2,C3.

c-summand:

(6)* $stop^{<}(t_b + \frac{10}{3})\, BFS(t_b + \frac{10}{3}, 0, move,$
$t_b + \frac{10}{3}, h - (3 - r)(t_b - t_c + \frac{10}{3}), inc)$
$\lhd s_b = nfill \wedge s_c = dec \wedge t_b - t_c < \tau_e - \frac{10}{3} \rhd \delta^{<}\mathbf{0}$

δ_{CB}-summand:

(7) $\delta^{<}(t_b + \frac{10}{3}) \lhd s_b = nfill \wedge s_c = dec \wedge t_b - t_c < \tau_e - \frac{10}{3} \rhd \delta^{<}\mathbf{0}$ ($\subseteq 6$)

δ_{Con}-summand:

(8) $\sum_{u:\mathbf{Time}} \delta^{<}u \lhd s_b = nfill \wedge s_c = dec \wedge u$
$\leqslant t_b + \frac{10}{3} \wedge u < t_c + \tau_e \rhd \delta^{<}\mathbf{0}$ ($\subseteq 6 + 13$)

CB2,C4.

δ_{CB}-summand:

(9) $\delta^{<}(t_b + \frac{10}{3}) \lhd s_b = nfill \wedge s_c = dec \wedge t_b - t_c \leqslant \tau_e - \frac{10}{3} \rhd \delta^{<}\mathbf{0}$ ($\subseteq 6 + 13$)

δ_{Con}-summand:

(10) $\delta^{<}(t_c + \tau_e) \lhd s_b = nfill \wedge s_c = dec \wedge t_c - t_b \leqslant \frac{10}{3} - \tau_e \rhd \delta^{<}\mathbf{0}$ ($\subseteq 13$)

CB3,C3.

δ_{CB}-summand:

(11) $\sum_{u:\mathbf{Time}} \sum_{t:\mathbf{Time}} \delta \lhd t$
$\quad \lhd s_b = nfill \wedge s_c = dec \wedge t \leqslant u \wedge t$
$\quad\quad \leqslant t_b + \frac{10}{3} \wedge u < t_c + \tau_e \rhd \delta \lhd \mathbf{0}$ $(\subseteq 6 + 13)$

δ_{Con}-summand:

(12) $\sum_{t:\mathbf{Time}} \sum_{u:\mathbf{Time}} \delta \lhd u$
$\quad \lhd s_b = nfill \wedge s_c = dec \wedge u \leqslant t \wedge t$
$\quad\quad \leqslant t_b + \frac{10}{3} \wedge u < t_c + \tau_e \rhd \delta \lhd \mathbf{0}$ $(\subseteq 6 + 13)$

CB3,C4.

c-summand:

(13)* $empty \lhd (t_c + \tau_e)\, BFS(t_c + \tau_e, 3(t_c - t_b + \tau_e), sfill, t_c + \tau_e, 0, dry)$
$\quad \lhd s_b = nfill \wedge s_c = dec \wedge t_c - t_b \leqslant \frac{10}{3} - \tau_e \rhd \delta \lhd \mathbf{0}$

δ_{CB}-summand:

(14) $\sum_{t:\mathbf{Time}} \delta \lhd t \lhd \ s_b = nfill \wedge s_c = dec \wedge t \leqslant t_b + \frac{10}{3} \wedge t$
$\quad \leqslant t_c + \tau_e \rhd \delta \lhd \mathbf{0}$ $(\subseteq 6 + 13)$

δ_{Con}-summand:

(15) $\delta \lhd (t_c + \tau_e) \lhd s_b = nfill \wedge s_c = dec \wedge t_c - t_b \leqslant \frac{10}{3} - \tau_e \rhd \delta \lhd \mathbf{0}$ $(\subseteq 13)$

CB4,C5.

c-summand:

(16)* $stop \lhd (t_b + \frac{10-l}{r})\, BFS(t_b + \frac{10-l}{r}, 0, move, t_b + \frac{10-l}{r}, 0, inc)$
$\quad \lhd s_b = sfill \wedge s_c = dry \rhd \delta \lhd \mathbf{0}$

δ_{CB}-summand:

(17) $\delta \lhd (t_b + \frac{10-l}{r}) \lhd s_b = sfill \wedge s_c = dry \rhd \delta \lhd \mathbf{0}$ $(\subseteq 16)$

δ_{Con}-summand:

(18) $\delta \lhd (t_b + \frac{10-l}{r}) \lhd s_b = sfill \wedge s_c = dry \rhd \delta \lhd \mathbf{0}$ $(\subseteq 16)$

Some elementary calculations show that only the summands marked with $*$ remain; the others can be eliminated. Behind the non-marked summands it is indicated by which marked summands they are absorbed. The resulting expression may be simplified further:

1. The time parameters t_b and t_c take on the same value in each non-vanishing summand. Therefore, the system can be characterised with a single time parameter t, which follows by an application of RSP.

2. The states $s_b = move \wedge s_c = inc$, $s_b = nfill \wedge s_c = dec$ and $s_b = sfill \wedge s_c = dry$ may be characterised by the natural numbers $1, 2$ and 3, respectively.
3. Process $\partial_H(CB(t_b, l, s_b) \| (t_c + \tau_f) \gg \delta^{\varsigma}(t_c + \tau_f))$ in summand (4) is easily proven equal to $\delta^{\varsigma}(t_c + \tau_f)$.

Consider the following process specification:

$$
\begin{aligned}
&\textbf{proc} \quad BFS'(t{:}\textbf{Time}, s{:}\mathbb{N}, l{:}\mathbb{R}, h{:}\mathbb{R}) = \\
&(1') \qquad start^{\varsigma}(t+1)\, BFS'(t+1, 2, 0, h+r) \\
&\qquad\qquad \lhd s = 1 \wedge 1 < \tau_f \rhd \delta^{\varsigma}\mathbf{0} + \\
&(4') \qquad overflow^{\varsigma}(t+\tau_f)\, \delta^{\varsigma}(t+\tau_f) \\
&\qquad\qquad \lhd s = 1 \wedge \tau_f \leqslant 1 \rhd \delta^{\varsigma}\mathbf{0} + \\
&(6') \qquad stop^{\varsigma}(t+\tfrac{10}{3})\, BFS'(t+\tfrac{10}{3}, 1, 0, h-\tfrac{10}{3}(3-r)) \\
&\qquad\qquad \lhd s = 2 \wedge \tfrac{10}{3} < \tau_e \rhd \delta^{\varsigma}\mathbf{0} + \\
&(13') \qquad empty^{\varsigma}(t+\tau_e)\, BFS'(t+\tau_e, 3, 3\tau_e, 0) \\
&\qquad\qquad \lhd s = 2 \wedge \tau_e \leqslant \tfrac{10}{3} \rhd \delta^{\varsigma}\mathbf{0} + \\
&(16') \qquad stop^{\varsigma}(t+\tfrac{10-l}{r})\, BFS'(t+\tfrac{10-l}{r}, 1, 0, 0) \\
&\qquad\qquad \lhd s = 3 \rhd \delta^{\varsigma}\mathbf{0}
\end{aligned}
$$

It follows by RSP that, provided that the invariant holds, $BFS(t, l, move, t, h, inc) = BFS'(t, 1, l, h)$.

3.3. Behaviour of the bottle filling system

We study the bottle filling system starting on time t, in state 1, with a half-full container. The capacity m of the container is chosen large enough (say $m > 10$) to guarantee normal behaviour, at least for some time. Process $BFS'(t, 1, l, m/2)$ can be analysed in detail, following three possible scenarios.

3.3.1. Optimal filling conditions

The ideal and most simple situation occurs when the contents h of the container always fluctuates around the same level $(m/2)$. It is easily found that this is the case when $r = \frac{30}{13}$. The bottle filling system then behaves according to the following equation:

$$BFS'(t, 1, l, \tfrac{m}{2}) = start^{\varsigma}(t+1)\, stop^{\varsigma}(t+\tfrac{13}{3})\, BFS'(t+\tfrac{13}{3}, 1, 0, \tfrac{m}{2}).$$

Using RSP this system can be simplified to

$$BFS_1(t) = start^{\varsigma}(t+1)\, stop^{\varsigma}(t+\tfrac{13}{3})\, BFS_1(t+\tfrac{13}{3}),$$

where $BFS'(t, 1, l, m/2) = BFS_1(t)$.

Note that it would even be possible to use a much smaller container and still have a well functioning bottle filling system. From the conditions in summands $(1')$ and $(6')$ of BFS' it follows easily that the ideal system works fine for all $m > 2r = \frac{60}{13}$.

3.3.2. Container overflow

Overflow occurs when $r > \frac{30}{13}$. First, we have the equation which describes how the container is getting fuller, until a moment just before overflow occurs. Note that it is quite similar to the equation for ideal behaviour.

For $h < m - r$ (this means that $1 < \tau_f$) no overflow occurs yet:

$$BFS'(t,1,l,h) = start^{<}(t+1)\,stop^{<}(t+\tfrac{13}{3})\,BFS'(t+\tfrac{13}{3},1,0,h+\tfrac{13}{3}(r-\tfrac{30}{13})).$$

We see, as $r > \frac{30}{13}$, that this system is not stable: The container contents h increases in time, and as long as $h' = h + \frac{13}{3}(r - \frac{30}{13}) < m - r$ no overfow occurs yet.

However, as soon as $h \geqslant m - r$ an overflow occurs and the system blocks:

$$BFS'(t,1,l,h) = overflow^{<}(t+\tau_f)\,\delta^{<}(t+\tau_f).$$

Using RSP we can easily prove that $BFS'(t,1,l,h)$ equals

proc $BFS_2(t,h) =$
$start^{<}(t+1)\,stop^{<}(t+\frac{13}{3})\,BFS_2(t+\frac{13}{3},h+\frac{13}{3}(r-\frac{30}{13})) \lhd 1 < \tau_f \rhd \delta^{<}\mathbf{0} +$
$overflow^{<}(t+\tau_f)\,\delta^{<}(t+\tau_f) \lhd \tau_f \leqslant 1 \rhd \delta^{<}\mathbf{0}$

3.3.3. Container underflow

Underflow occurs when $r < \frac{30}{13}$. First, we have the equation which describes how the container is getting emptier (a), until a moment just before it gets totally empty (b).

- For $h > 10 - \frac{13}{3}r$ (this means that $\frac{10}{3} < \tau_e$) we have that

  (a) $BFS'(t,1,l,h) = start^{<}(t+1)\,stop^{<}(t+\frac{13}{3})\,BFS'(t+\frac{13}{3},1,0,h-\frac{13}{3}(\frac{30}{13}-r))$.

  Here we see that the container contents decreases in time. The following steps – specified by (a), (b) or (c) – depend on the value of $h' = h - \frac{13}{3}(\frac{30}{13} - r)$.
- For $h \leqslant 10 - \frac{13}{3}r$ it follows that

  (b) $BFS'(t,1,l,h) =$
  $start^{<}(t+1)\,empty^{<}(t+\frac{3}{3-r}+\tau_e)\,stop^{<}(t+\frac{10}{r}+\tau_e(1-\frac{3}{r}))$
  $BFS'(t+\frac{10}{r}+\tau_e(1-\frac{3}{r}),1,0,0)$.

  Finally, let

  proc $BFS_3(t{:}\mathbf{Time}) = start^{<}(t+1)\,empty^{<}(t+\frac{3}{3-r})\,stop^{<}(t+\frac{10}{r})\,BFS_3(t+\frac{10}{r})$

  Using RSP it easily follows that

  (c) $BFS'(t,1,0,0) = BFS_3(t)$.

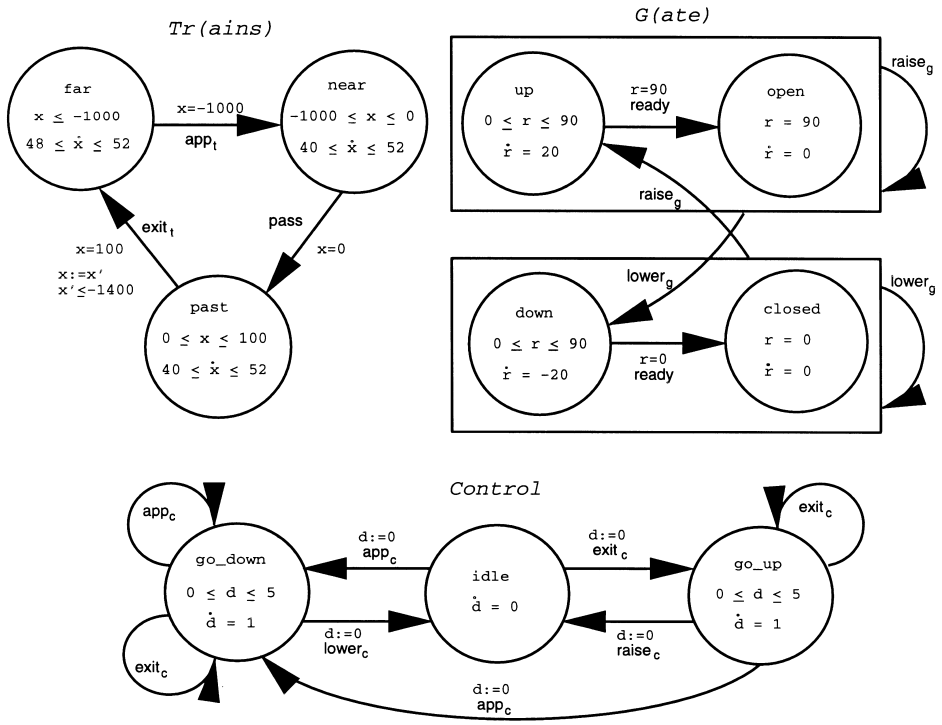

Fig. 4. The components of the railroad gate controller.

We see that the process under (a), the most general case, converges to (b), which in turn evolves to (c). During the filling of each bottle the container gets empty, so that the filling process slows down. Note that when r gets closer to $\frac{30}{13}$, the moments on which the *empty* and *stop* actions happen both move closer to $t + \frac{13}{3}$.

4. A railroad gate controller

4.1. Specification

The following example is about a hybrid control system for a railroad crossing. It originates from [2]. Three processes are involved: *Tr*(*ains*), *G*(*ate*) and *Control*. Schematically, the processes can be represented as in Fig. 4.

The figure is taken from [2]. State transitions of components are denoted by arrows from one state to another. In the picture of the *G*(*ate*) process transitions between boxes denote transitions to and from all states in the boxes concerned. E.g., the action $lower_g$ changes the states with *down* and *closed* to themselves. The components communicate by the subscripted actions. Moreover, there are two different autonomous transitions, i.e., the passing of the train (*pass*) and the completion of opening and closing the gate (*ready*).

The Tr process is specified by the equation below:

$$
\begin{aligned}
&\textbf{proc} \quad Tr(t_t{:}\textbf{Time}, s_t{:}TState) = \\
&(\mathrm{Tr1}) \quad \overline{\sum}_{f:Func,t:\textbf{Time}} app_t \vartriangleleft t\, Tr(t, near) \\
&\qquad \triangleleft s_t = far \wedge f(t_t) \leqslant -1400 \wedge f(t) = -1000 \wedge \forall t'.48 \leqslant f'(t') \leqslant 52 \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Tr2}) \quad \overline{\sum}_{f:Func,t:\textbf{Time}} pass \vartriangleleft t\, Tr(t, past) \\
&\qquad \triangleleft s_t = near \wedge f(t_t) = -1000 \wedge f(t) = 0 \wedge \forall t'.40 \leqslant f'(t') \leqslant 52 \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Tr3}) \quad \overline{\sum}_{f:Func,t:\textbf{Time}} exit_t \vartriangleleft t\, Tr(t, far) \\
&\qquad \triangleleft s_t = past \wedge f(t_t) = 0 \wedge f(t) = 100 \wedge \forall t'.40 \leqslant f'(t') \leqslant 52 \triangleright \delta \vartriangleleft \mathbf{0}
\end{aligned}
$$

When a train approaches the gate from a great distance ($\leqslant -1000\,\mathrm{m}$) it has a velocity $48 \leqslant \dot{x} \leqslant 52\,\mathrm{m/s}$. As soon as it passes a detector placed at $-1000\,\mathrm{m}$ a signal app_t is sent to the controller (Tr1). The train may now slow down according to the inequality $40 \leqslant \dot{x} \leqslant 52\,\mathrm{m/s}$, and pass the gate (Tr2). After 100 m another detector signals $exit_t$ to the controller (Tr3). A new train may come after the current one has passed the second detector, but only at a distance $\geqslant 1500\,\mathrm{m}$.

The gate's signals $lower_g$ and $raise_g$ are driven by the controller. The gate lowers from 90° to 0° at a constant rate of 20°/s, and it raises from 0° to 90° at the same rate. The gate must always accept controller commands.

$$
\begin{aligned}
&\textbf{proc} \quad G(t_g{:}\textbf{Time}, s_g{:}GState, r{:}\mathbb{R}) = \\
&(\mathrm{Ga1}) \quad \sum_{u:\textbf{Time}} lower_g \vartriangleleft u\, G(u, down, 90) \\
&\qquad \triangleleft s_g = open \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Ga2}) \quad \sum_{u:\textbf{Time}} lower_g \vartriangleleft u\, G(u, closed, 0) \\
&\qquad \triangleleft s_g = closed \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Ga3}) \quad \overline{\sum}_{f:Func,u:\textbf{Time}} lower_g \vartriangleleft u\, G(u, down, f(u)) \\
&\qquad \triangleleft s_g = up \wedge f(t_g) = r \wedge f(u) \leqslant 90 \wedge \forall t.f'(t) = 20 \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Ga4}) \quad \overline{\sum}_{f:Func,u:\textbf{Time}} lower_g \vartriangleleft u\, G(u, down, f(u)) \\
&\qquad \triangleleft s_g = down \wedge f(t_g) = r \wedge 0 \leqslant f(u) \wedge \forall t.f'(t) = -20 \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Ga5}) \quad \overline{\sum}_{f:Func,u:\textbf{Time}} ready \vartriangleleft u\, G(u, closed, 0) \\
&\qquad \triangleleft s_g = down \wedge f(t_g) = r \wedge 0 = f(u) \wedge \forall t.f'(t) = -20 \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Ga6}) \quad \sum_{u:\textbf{Time}} raise_g \vartriangleleft u\, G(u, up, 0) \\
&\qquad \triangleleft s_g = closed \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Ga7}) \quad \sum_{u:\textbf{Time}} raise_g \vartriangleleft u\, G(u, open, 90) \\
&\qquad \triangleleft s_g = open \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Ga8}) \quad \overline{\sum}_{f:Func,u:\textbf{Time}} raise_g \vartriangleleft u\, G(u, up, f(u)) \\
&\qquad \triangleleft s_g = up \wedge f(t_g) = r \wedge f(u) \leqslant 90 \wedge \forall t.f'(t) = 20 \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Ga9}) \quad \overline{\sum}_{f:Func,u:\textbf{Time}} raise_g \vartriangleleft u\, G(u, up, f(u)) \\
&\qquad \triangleleft s_g = down \wedge f(t_g) = r \wedge 0 \leqslant f(u) \wedge \forall t.f'(t) = -20 \triangleright \delta \vartriangleleft \mathbf{0} + \\
&(\mathrm{Ga10}) \quad \overline{\sum}_{f:Func,u:\textbf{Time}} ready \vartriangleleft u\, G(u, open, 90) \\
&\qquad \triangleleft s_g = up \wedge f(t_g) = r \wedge f(u) = 90 \wedge \forall t.f'(t) = 20 \triangleright \delta \vartriangleleft \mathbf{0}
\end{aligned}
$$

The controller is driven by train detector signals app_t and $exit_t$, and it should be able to receive these at any time. After an app_t signal has been issued, it takes the controller at most 5 s to send the command $lower_c$ to the gate. After receiving an $exit_t$ signals it takes at most 5 s to send a $raise_c$ signal to the gate.

Fault tolerance considerations prescribe that $exit_t$ signals should always be ignored if the gate is about to be lowered, and that app_t signals always should cause the gate to go down. The controller process uses delay d:**Time** to keep track of how long it has been preparing already for sending a message. State *go_up* denotes the state where the controller is bound to send a $raise_c$ signal, and in *go_down* the controller is bound to send a $lower_c$ signal.

$$\begin{aligned}
&\textbf{proc} \quad Control(t_c{:}\textbf{Time}, s_c{:}CState, d{:}\textbf{Time}) =\\
&(C1)\quad \textstyle\sum_{v:\textbf{Time}} app_c{?}v\ Control(v, go_down, d+v-t_c)\\
&\qquad\qquad \lhd s_c = go_down \wedge t_c \leqslant v \leqslant t_c + 5 - d \rhd \delta \lhd \mathbf{0} +\\
&(C2)\quad \textstyle\sum_{v:\textbf{Time}} app_c{?}v\ Control(v, go_down, \mathbf{0})\\
&\qquad\qquad \lhd s_c = go_up \wedge t_c \leqslant v \leqslant t_c + 5 - d \rhd \delta \lhd \mathbf{0} +\\
&(C3)\quad \textstyle\sum_{v:\textbf{Time}} app_c{?}v\ Control(v, go_down, \mathbf{0})\\
&\qquad\qquad \lhd s_c = idle \wedge t_c \leqslant v \rhd \delta \lhd \mathbf{0} +\\
&(C4)\quad \textstyle\sum_{v:\textbf{Time}} exit_c{?}v\ Control(v, go_up, d+v-t_c)\\
&\qquad\qquad \lhd s_c = go_up \wedge t_c \leqslant v \leqslant t_c + 5 - d \rhd \delta \lhd \mathbf{0} +\\
&(C5)\quad \textstyle\sum_{v:\textbf{Time}} exit_c{?}v\ Control(v, go_up, \mathbf{0})\\
&\qquad\qquad \lhd s_c = idle \wedge t_c \leqslant v \rhd \delta \lhd \mathbf{0} +\\
&(C6)\quad \textstyle\sum_{v:\textbf{Time}} exit_c{?}v\ Control(v, go_down, d+v-t_c)\\
&\qquad\qquad \lhd s_c = go_down \wedge t_c \leqslant v \leqslant t_c + 5 - d \rhd \delta \lhd \mathbf{0} +\\
&(C7)\quad \textstyle\sum_{v:\textbf{Time}} raise_c{?}v\ Control(v, idle, \mathbf{0})\\
&\qquad\qquad \lhd s_c = go_up \wedge t_c \leqslant v \leqslant t_c + 5 - d \rhd \delta \lhd \mathbf{0} +\\
&(C8)\quad \textstyle\sum_{v:\textbf{Time}} lower_c{?}v\ Control(v, idle, \mathbf{0})\\
&\qquad\qquad \lhd s_c = go_down \wedge t_c \leqslant v \leqslant t_c + 5 - d \rhd \delta \lhd \mathbf{0}
\end{aligned}$$

4.2. Simplification of the components

The conditions in the *Tr* and *G* processes may be simplified, because upper and lower bounds for the values of the time parameters t and u, respectively, can be derived. After some elementary manipulations we obtain the process *Trains*: (We will not go into the details of the calculations.)

$$\begin{aligned}
&\textbf{proc} \quad Trains(t_t{:}\textbf{Time}, s_t{:}TState) =\\
&(T1)\quad \textstyle\sum_{t:\textbf{Time}} app_t{?}t\ Trains(t, near)\\
&\qquad\qquad \lhd s_t = far \wedge t_t + \tfrac{400}{52} \leqslant t \rhd \delta \lhd \mathbf{0} +\\
&(T2)\quad \textstyle\sum_{t:\textbf{Time}} pass{?}t\ Trains(t, past)\\
&\qquad\qquad \lhd s_t = near \wedge t_t + \tfrac{1000}{52} \leqslant t \leqslant t_t + 25 \rhd \delta \lhd \mathbf{0} +\\
&(T3)\quad \textstyle\sum_{t:\textbf{Time}} exit_t{?}t\ Trains(t, far)\\
&\qquad\qquad \lhd s_t = past \wedge t_t + \tfrac{100}{52} \leqslant t \leqslant t_t + \tfrac{10}{4} \rhd \delta \lhd \mathbf{0}
\end{aligned}$$

In a similar way, a reduced specification for the gate process can be derived:

$$
\begin{aligned}
&\textbf{proc} \quad Gate(t_g{:}\textbf{Time}, s_g{:}GState, r{:}\mathbb{R}) = \\
&(\text{G1}) \quad \textstyle\sum_{u:\textbf{Time}} lower_g \vartriangleleft u \mathbin{\vartriangleleft} Gate(u, down, 90) \\
&\qquad\qquad \vartriangleleft s_g = open \vartriangleright \delta \vartriangleleft \mathbf{0} + \\
&(\text{G2}) \quad \textstyle\sum_{u:\textbf{Time}} lower_g \vartriangleleft u \mathbin{\vartriangleleft} Gate(u, closed, 0) \\
&\qquad\qquad \vartriangleleft s_g = closed \vartriangleright \delta \vartriangleleft \mathbf{0} + \\
&(\text{G3}) \quad \textstyle\sum_{u:\textbf{Time}} lower_g \vartriangleleft u \mathbin{\vartriangleleft} Gate(u, down, 20(u - t_g) + r) \\
&\qquad\qquad \vartriangleleft s_g = up \wedge u \leqslant t_g + \tfrac{90-r}{20} \vartriangleright \delta \vartriangleleft \mathbf{0} + \\
&(\text{G4}) \quad \textstyle\sum_{u:\textbf{Time}} lower_g \vartriangleleft u \mathbin{\vartriangleleft} Gate(u, down, 20(t_g - u) + r) \\
&\qquad\qquad \vartriangleleft s_g = down \wedge u \leqslant t_g + \tfrac{r}{20} \vartriangleright \delta \vartriangleleft \mathbf{0} + \\
&(\text{G5}) \quad ready \vartriangleleft (t_g + \tfrac{r}{20}) \mathbin{\vartriangleleft} Gate(t_g + \tfrac{r}{20}, closed, 0) \\
&\qquad\qquad \vartriangleleft s_g = down \vartriangleright \delta \vartriangleleft \mathbf{0} + \\
&(\text{G6}) \quad \textstyle\sum_{u:\textbf{Time}} raise_g \vartriangleleft u \mathbin{\vartriangleleft} Gate(u, up, 0) \\
&\qquad\qquad \vartriangleleft s_g = closed \vartriangleright \delta \vartriangleleft \mathbf{0} + \\
&(\text{G7}) \quad \textstyle\sum_{u:\textbf{Time}} raise_g \vartriangleleft u \mathbin{\vartriangleleft} Gate(u, open, 90) \\
&\qquad\qquad \vartriangleleft s_g = open \vartriangleright \delta \vartriangleleft \mathbf{0} + \\
&(\text{G8}) \quad \textstyle\sum_{u:\textbf{Time}} raise_g \vartriangleleft u \mathbin{\vartriangleleft} Gate(u, up, 20(u - t_g) + r) \\
&\qquad\qquad \vartriangleleft s_g = up \wedge u \leqslant t_g + \tfrac{90-r}{20} \vartriangleright \delta \vartriangleleft \mathbf{0} + \\
&(\text{G9}) \quad \textstyle\sum_{u:\textbf{Time}} raise_g \vartriangleleft u \mathbin{\vartriangleleft} Gate(u, up, 20(t_g - u) + r) \\
&\qquad\qquad \vartriangleleft s_g = down \wedge u \leqslant t_g + \tfrac{r}{20} \vartriangleright \delta \vartriangleleft \mathbf{0} + \\
&(\text{G10}) \quad ready \vartriangleleft (t_g + \tfrac{90-r}{20}) \mathbin{\vartriangleleft} Gate(t_g + \tfrac{90-r}{20}, open, 90) \\
&\qquad\qquad \vartriangleleft s_g = up \vartriangleright \delta \vartriangleleft \mathbf{0}
\end{aligned}
$$

Let

$$
\begin{aligned}
H_1 &\stackrel{\text{def}}{=} \{app_t, app_c, exit_t, exit_c\} \\
H_2 &\stackrel{\text{def}}{=} \{raise_g, raise_c, lower_g, lower_c\}
\end{aligned}
$$

and communications be defined by

$$
\begin{aligned}
&\textbf{comm} \quad app_t \mid app_c = app \\
&\qquad\quad exit_t \mid exit_c = exit \\
&\qquad\quad raise_g \mid raise_c = raise \\
&\qquad\quad lower_g \mid lower_c = lower
\end{aligned}
$$

In order to make a modular analysis of the complete system, we split the specification in two parts. One module contains the trains process and the controller, and the other module contains the first module together with the gate process. The total system can now be described by

$$
\begin{aligned}
&\textbf{proc} \quad TC(t_t{:}\textbf{Time}, s_t{:}TState, t_c{:}\textbf{Time}, s_c{:}CState, d{:}\textbf{Time}) \\
&\qquad\quad = \partial_{H_1}(Trains(t_t, s_t) \parallel Control(t_c, s_c, d))
\end{aligned}
$$

$$RGC(t_t{:}\mathbf{Time}, s_t{:}TState, t_c{:}\mathbf{Time}, s_c{:}CState, d{:}\mathbf{Time}, t_g{:}\mathbf{Time}, s_g{:}GState, r{:}\mathbb{R})$$
$$= \partial_{H_2}(TC(t_t, s_t, t_c, s_c, d) \,\|\, Gate(t_g, s_g, r))$$

4.3. Expansion and analysis of process TC

The first step in the linearisation of the railroad gate controller process is the linearisation of the system module $\partial_{H_1}(Trains(t_t, s_t) \,\|\, Control(t_c, s_c, d))$.

4.3.1. Encapsulation

In a similar way as in Section 3.2 we have to start by expanding and encapsulating the equation for TC, according to Theorem B.2. For this purpose, we identify p with *Trains* and q with *Control*. Five different Δ-summands are distinguished.

First, we only consider the non-δ summands, namely $\Delta 1, \Delta 2, \Delta 3$:

$\Delta 1$ consists of the c-summands (communications between *Trains* and *Control*):

$$\partial_{H_1}(T1 \mid C1), \partial_{H_1}(T1 \mid C2), \partial_{H_1}(T1 \mid C3), \partial_{H_1}(T3 \mid C4), \partial_{H_1}(T3 \mid C5), \partial_{H_1}(T3 \mid C6);$$

$\Delta 2$ consists of the autonomous *Trains*-summands:

$$\partial_{H_1}(T2 \lfloor\!\lfloor C1), \ldots, \partial_{H_1}(T2 \lfloor\!\lfloor C8);$$

$\Delta 3$ consists of the autonomous *Control*-summands:

$$\partial_{H_1}(C7 \lfloor\!\lfloor T1), \partial_{H_1}(C7 \lfloor\!\lfloor T2), \partial_{H_1}(C7 \lfloor\!\lfloor T3), \partial_{H_1}(C8 \lfloor\!\lfloor T1), \partial_{H_1}(C8 \lfloor\!\lfloor T2), \partial_{H_1}(C8 \lfloor\!\lfloor T3).$$

Expansion of the various terms is straightforward. It leads to the following set of terms:

C-summands:

T1,C1.

(TC1) $\sum_{t:\mathbf{Time}} app{<}t\, TC(t, near, t, go\_down, d + t - t_c)$
$\quad \lhd s_t = far \wedge s_c = go\_down \wedge max(t_t + \frac{400}{52}, t_c) \leqslant t \leqslant t_c + 5 - d \rhd \delta{<}\mathbf{0}$

T1,C2.

(TC2) $\sum_{t:\mathbf{Time}} app{<}t\, TC(t, near, t, go\_down, \mathbf{0})$
$\quad \lhd s_t = far \wedge s_c = go\_up \wedge max(t_t + \frac{400}{52}, t_c) \leqslant t \leqslant t_c + 5 - d \rhd \delta{<}\mathbf{0}$

T1,C3.

(TC3) $\sum_{t:\mathbf{Time}} app{<}t\, TC(t, near, t, go\_down, \mathbf{0})$
$\quad \lhd s_t = far \wedge s_c = idle \wedge max(t_t + \frac{400}{52}, t_c) \leqslant t \rhd \delta{<}\mathbf{0}$

T3,C4.

(TC4) $\sum_{t:\mathbf{Time}} exit{<}t\, TC(t, far, t, go\_up, d + t - t_c)$
$\quad \lhd s_t = past \wedge s_c = go\_up \wedge max(t_t + \frac{100}{52}, t_c) \leqslant t$
$\quad\quad \leqslant min(t_t + \frac{10}{4}, t_c + 5 - d)\delta{<}\mathbf{0}$

T3,C5.

(TC5) $\sum_{t:\mathbf{Time}} exit{<}t\, TC(t, far, t, go\_up, \mathbf{0})$
$\quad \lhd s_t = past \wedge s_c = idle \wedge max(t_t + \frac{100}{52}, t_c) \leqslant t \leqslant t_t + \frac{10}{4} \rhd \delta{<}\mathbf{0}$

T3,C6.

(TC6) $\sum_{t:\mathbf{Time}} exit \lhd t \; TC(t, far, t, go\_down, d + t - t_c)$
$\quad \lhd s_t = past \wedge s_c = go\_down \wedge max(t_t + \frac{100}{52}, t_c) \leqslant t$
$\quad \leqslant min(t_t + \frac{10}{4}, t_c + 5 - d) \rhd \delta \lhd \mathbf{0}$

Autonomous *Trains*-**summands:**

T2,C{3, 5}.

(TC7) $\sum_{t:\mathbf{Time}} pass \lhd t \; TC(t, past, t_c, idle, d)$
$\quad \lhd s_t = near \wedge s_c = idle \wedge max(t_t + \frac{1000}{52}, t_c) \leqslant t \leqslant t_t + 25 \rhd \delta \lhd \mathbf{0}$

T2,C{1, 2, 4, 6, 7, 8}.

(TC8) $\sum_{t:\mathbf{Time}} pass \lhd t \; TC(t, past, t_c, s_c, d)$
$\quad \lhd s_t = near \wedge s_c \neq idle \wedge max(t_t + \frac{1000}{52}, t_c) \leqslant t$
$\quad \leqslant min(t_t + 25, t_c + 5 - d) \rhd \delta \lhd \mathbf{0}$

Autonomous *Control*-**summands:**

T1,C7.

(TC9) $\sum_{v:\mathbf{Time}} raise_c \lhd v \; TC(t_t, far, v, idle, \mathbf{0})$
$\quad \lhd s_t = far \wedge s_c = go\_up \wedge t_c \leqslant v \leqslant t_c + 5 - d \rhd \delta \lhd \mathbf{0}$

T2,C7.

(TC10) $\sum_{v:\mathbf{Time}} raise_c \lhd v \; TC(t_t, near, v, idle, \mathbf{0})$
$\quad \lhd s_t = near \wedge s_c = go\_up \wedge t_c \leqslant v \leqslant min(t_t + 25, t_c + 5 - d) \rhd \delta \lhd \mathbf{0}$

T3,C7.

(TC11) $\sum_{v:\mathbf{Time}} raise_c \lhd v \; TC(t_t, past, v, idle, \mathbf{0})$
$\quad \lhd s_t = past \wedge s_c = go\_up \wedge t_c \leqslant v \leqslant min(t_t + \frac{10}{4}, t_c + 5 - d) \rhd \delta \lhd \mathbf{0}$

T1,C8.

(TC12) $\sum_{v:\mathbf{Time}} lower_c \lhd v \; TC(t_t, far, v, idle, \mathbf{0})$
$\quad \lhd s_t = far \wedge s_c = go\_down \wedge t_c \leqslant v \leqslant t_c + 5 - d \rhd \delta \lhd \mathbf{0}$

T2,C8.

(TC13) $\sum_{v:\mathbf{Time}} lower_c \lhd v \; TC(t_t, near, v, idle, \mathbf{0})$
$\quad \lhd s_t = near \wedge s_c = go\_down \wedge t_c \leqslant v \leqslant min(t_t + 25, t_c + 5 - d) \rhd \delta \lhd \mathbf{0}$

T3,C8.

(TC14) $\sum_{v:\mathbf{Time}} lower_c \lhd v \; TC(t_t, past, v, idle, \mathbf{0})$
$\quad \lhd s_t = past \wedge s_c = go\_down \wedge t_c \leqslant v \leqslant min(t_t + \frac{10}{4}, t_c + 5 - d) \rhd \delta \lhd \mathbf{0}$

Note that we already made two more steps:

1. All eight autonomous *Trains*-summands are combined in only two summands;
2. The conditions of the autonomous summands are simplified.

These manipulations are quite elementary, and therefore not treated in detail.

The proof is still not complete yet, since the Encapsulation Theorem shows that there are two more main summands to be dealt with: $\Delta 4$ and $\Delta 5$.

$\Delta 4$ consists of the δ_{Trains}-summands:

$$\partial_{H_1}(T1 \between C1), \ldots, \partial_{H_1}(T1 \between C8), \partial_{H_1}(T3 \between C1), \ldots, \partial_{H_1}(T3 \between C8);$$

$\Delta 5$ consists of the $\delta_{Control}$-summands:

$$\partial_{H_1}(C_i \between T_j), \quad \text{where } i \in \{1,2,3,4,5,6\}, \ \ j \in \{1,2,3\}.$$

Now all these terms can be eliminated from TC. The way to do so, is in principle based on the identity $a \lhd t\, x + \delta \lhd t = a \lhd t\, x$, which can easily be derived from the axioms of $\mu\mathrm{CRL}_t$. So elimination of a term with time deadlocks, such as the terms mentioned above, boils down to a proof that it is *included* in an autonomous or c-summand. This job (note that there are 34 such terms) would be quite trying if there was no easier way to get rid of most of them. Fortunately, the elimination of time deadlocks turns out to be much easier using an invariant.

4.3.2. An invariant

Starting from the assumption that initially the train is far away and the gates are open, it is not difficult to formulate an invariant.

Let $I_{TC}(s_t, s_c, t_t, t_c)$ be defined by

$$\begin{aligned}&(s_t = far \wedge s_c = go\_up \wedge t_t = t_c) \vee (s_t = far \wedge s_t = idle \wedge t_c \leqslant t_t + 5 - d)\\ &\quad \vee (s_t = past \wedge s_c = idle \wedge t_c \leqslant t_t) \vee (s_t = near \wedge s_c = idle \wedge t_c \leqslant t_t + 5 - d)\\ &\quad \vee (s_t = near \wedge s_c = go\_down \wedge t_t = t_c).\end{aligned}$$

It is easily verified that I_{TC} is an invariant for TC. Note that for a correctness proof of the invariant we do not have to take $\Delta 4$ and $\Delta 5$ into account: Deadlocks do not represent actions, and therefore do not cause any state transitions.

Now that we have this invariant at our disposal, the majority of the time deadlocks from $\Delta 4$ and $\Delta 5$ may be eliminated, because, provided that the invariant holds, the conditions belonging to most of the time deadlocks considered never become true. After this reduction, only a handful of time deadlocks are left to eliminate in the (equational) way sketched above.

So, provided that this invariant holds, TC consists of the terms TC1–TC14. But, using the invariant, TC may even be reduced further: The summands TC$\{1,2,4,6,8,10,11,12,14\}$ are cancelled, and the remaining summands may be rewritten using the corresponding inequalities in the invariant. Now we can also observe that parameter t_c plays no role any more in conditions or in time labels attached to actions. Therefore it may be eliminated.

The resulting system is given by

$$I_{TC}(s_t, s_c, t_t, t_c) \to TC(t_t, s_t, t_c, s_c, d) = TC'(t_t, s_t, s_c, d),$$

where

$$\begin{aligned}
&\textbf{proc} \quad TC'(t_t{:}\textbf{Time}, s_t{:}TState, s_c{:}CState, d{:}\textbf{Time}) =\\
&(\mathrm{TC3}') \quad \textstyle\sum_{t:\textbf{Time}} app{<}t\ TC'(t, near, go_down, \mathbf{0})\\
&\qquad\qquad \lhd s_t = far \wedge s_c = idle \wedge t_t + \tfrac{400}{52} \leqslant t \rhd \delta{<}\mathbf{0}+\\
&(\mathrm{TC5}') \quad \textstyle\sum_{t:\textbf{Time}} exit{<}t\ TC'(t, far, go_up, \mathbf{0})\\
&\qquad\qquad \lhd s_t = past \wedge s_c = idle \wedge t_t + \tfrac{100}{52} \leqslant t \leqslant t_t + \tfrac{10}{4} \rhd \delta{<}\mathbf{0}+\\
&(\mathrm{TC7}') \quad \textstyle\sum_{t:\textbf{Time}} pass{<}t\ TC'(t, past, idle, d)\\
&\qquad\qquad \lhd s_t = near \wedge s_c = idle \wedge t_t + \tfrac{1000}{52} \leqslant t \leqslant t_t + 25 \rhd \delta{<}\mathbf{0}+\\
&(\mathrm{TC9}') \quad \textstyle\sum_{v:\textbf{Time}} raise_c{<}v\ TC'(t_t, far, idle, \mathbf{0})\\
&\qquad\qquad \lhd s_t = far \wedge s_c = go_up \wedge t_t \leqslant v \leqslant t_t + 5 - d \rhd \delta{<}\mathbf{0}+\\
&(\mathrm{TC13}') \quad \textstyle\sum_{v:\textbf{Time}} lower_c{<}v\ TC'(t_t, near, idle, \mathbf{0})\\
&\qquad\qquad \lhd s_t = near \wedge s_c = go_down \wedge t_t \leqslant v \leqslant t_t + 5 - d \rhd \delta{<}\mathbf{0}
\end{aligned}$$

4.4. A linearised variant of the railroad gate controller

The following step in the analysis of the railroad gate controller is to expand and analyse the process $RGC(t_t, s_t, t_c, s_c, d, t_g, s_g, r) = \partial_{H_2}(TC(t_t, s_t, t_c, s_c, d) \,\|\, Gate(t_g, s_g, r))$, as specified in Section 4.2, using the equation for process *Gate* and the linear expression just derived for TC'.

4.4.1. Encapsulation

In order to provide the reader with a good understanding of the complexity of the analysis, we first give the various Δ-summands a straightforward application of Theorem B.2 would yield. We identify p with TC' and q with *Gate*.

Again, five different Δ-summands are distinguished:

$\Delta 1$ consists of the c-summands (communications between TC' and *Gate*):

$\partial_{H_2}(TC9' \mid G_i)$, where $i \in \{6,7,8,9\}$,

$\partial_{H_2}(TC13' \mid G_i)$, where $i \in \{1,2,3,4\}$;

$\Delta 2$ consists of the autonomous TC'-summands:

$\partial_{H_2}(TC'_i \lfloor\!\lfloor G_j)$, where $i \in \{3,5,7\}$, $j \in \{1,\ldots,10\}$;

$\Delta 3$ consists of the autonomous *Gate*-summands:

$\partial_{H_2}(G_i \lfloor\!\lfloor TC'_j)$, where $i \in \{5,10\}$, $j \in \{3,5,7,9,13\}$;

$\Delta 4$ consists of the $\delta_{TC'}$-summands:

$\partial_{H_2}(TC'_i \lfloor\!\lfloor G_j)$, where $i \in \{9,13\}$, $j \in \{1,\ldots,10\}$;

$\Delta 5$ consists of the δ_{Gate}-summands:

$\partial_{H_2}(G_i \lfloor\!\lfloor TC'_j)$, where $i \in \{1,2,3,4,6,7,8,9\}$, $j \in \{3,5,7,9,13\}$.

So in principle, there are 108 main terms to analyse, 60 of which consist of time deadlocks. Fortunately, there are quite easy ways to get rid of a lot of irrelevant terms.

4.4.2. A reachability analysis using a simple invariant

In order to simplify our analysis, we combine the state variables s_t, s_c and s_g in a tuple $s = \langle s_t, s_c, s_g \rangle$. As a first step in the analysis we may regard each possible action of *RGC* as a transformation of tuple $\langle s_t, s_c, s_g \rangle$ to a tuple $\langle s_t', s_c', s_g' \rangle$, and discard the other conditions. All possible transformations between tuples can be combined in a directed graph that has tuples as nodes and actions as transition labels.

Starting from initial state $\langle far, idle, open \rangle$ we come – via the autonomous TC'- and *Gate*-summands and communications – across the following states:

$\sigma_1 : \langle far, idle, open \rangle$

$\sigma_2 : \langle near, go\_down, open \rangle$

$\sigma_3 : \langle near, idle, down \rangle$

$\sigma_4 : \langle near, idle, closed \rangle$

$\sigma_5 : \langle past, idle, closed \rangle$

$\sigma_6 : \langle far, go\_up, closed \rangle$

$\sigma_7 : \langle far, idle, up \rangle$

$\sigma_8 : \langle near, go\_down, up \rangle$

$\sigma_9 : \langle past, idle, down \rangle$

$\sigma_{10} : \langle far, go\_up, down \rangle$

We can use this knowledge for a formal approach; provided that the condition $\bigvee_{i=1\ldots10} s = \sigma_i$ holds (and, of course, $I_{TC}(s_t, s_c, t_t, t_c)$), process *RGC* is equal to RGC', where RGC' satisfies the recursion equation below. Without proof we state that the δ-summands ($\Delta 4$ and $\Delta 5$) are all cancelled right away. Thirty time deadlocks may be cancelled using $\bigvee_{i=1\ldots10} s = \sigma_i$. The other 30 have to be considered separately.

For clarity, we first give the equation for RGC' only by reference to the main summands of TC' and *Gate*:

proc $RGC'(s{:}RState, t_t{:}\mathbf{Time}, t_g{:}\mathbf{Time}, d{:}\mathbf{Time}, r{:}\mathbb{R}) =$

(1) $\partial_{H_2}(TC3' \between G1) + \partial_{H_2}(TC3' \between G7) +$

(2) $\partial_{H_2}(TC13' \mid G1) +$

(3) $\partial_{H_2}(G5 \between TC7') +$

(4) $\partial_{H_2}(TC7' \between G2) + \partial_{H_2}(TC7' \between G6) +$

(5) $\partial_{H_2}(TC5' \between G2) + \partial_{H_2}(TC5' \between G6) +$

(6) $\partial_{H_2}(TC9' \mid G6) +$

(7) $\partial_{H_2}(G10 \between TC3') +$

(8) $\partial_{H_2}(TC3' \between G3) + \partial_{H_2}(TC3' \between G8) + \partial_{H_2}(TC3' \between G10) +$

(9) $\partial_{H_2}(G10 \between TC13') +$

(10) $\partial_{H_2}(TC13' \mid G3) +$

(11) $\partial_{H_2}(TC7' \between G4) + \partial_{H_2}(TC7' \between G5) + \partial_{H_2}(TC7' \between G9) +$

(12) $\partial_{H_2}(G5 \between TC5') +$

(13) $\partial_{H_2}(TC5' \between G4) + \partial_{H_2}(TC5' \between G5) + \partial_{H_2}(TC5' \between G9) +$

(14) $\partial_{H_2}(G5 \between TC9') +$

(15) $\partial_{H_2}(TC9' \mid G9)$

After quite some elementary calculations we find an expanded equation for RGC':

proc $RGC'(s{:}RState, t_t{:}\mathbf{Time}, t_g{:}\mathbf{Time}, d{:}\mathbf{Time}, r{:}\mathbb{R}) =$

(1) $\sum_{t:\mathbf{Time}} app \lhd t \rhd RGC'(2, t, t_g, \mathbf{0}, r)$
$\quad \lhd s = 1 \wedge t_t + \frac{400}{52} \leqslant t \rhd \delta \lhd \mathbf{0} +$

(2) $\sum_{u:\mathbf{Time}} lower \lhd u \rhd RGC'(3, t_t, u, \mathbf{0}, 90)$
$\quad \lhd s = 2 \wedge t_t \leqslant u \leqslant t_t + 5 - d \rhd \delta \lhd \mathbf{0} +$

(3) $ready \lhd (t_g + \frac{r}{20}) \rhd RGC'(4, t_t, t_g + \frac{r}{20}, d, 0)$
$\quad \lhd s = 3 \wedge max(t_g + \frac{r}{20}, t_t + \frac{1000}{52}) \leqslant t_t + 25 \rhd \delta \lhd \mathbf{0} +$

(4) $\sum_{t:\mathbf{Time}} pass \lhd t \rhd RGC'(5, t, t_g, d, r)$
$\quad \lhd s = 4 \wedge t_t + \frac{1000}{52} \leqslant t \leqslant t_t + 25 \rhd \delta \lhd \mathbf{0} +$

(5) $\sum_{t:\mathbf{Time}} exit \lhd t \rhd RGC'(6, t, t_g, \mathbf{0}, r)$
$\quad \lhd s = 5 \wedge t_t + \frac{100}{52} \leqslant t \leqslant t_t + \frac{10}{4} \rhd \delta \lhd \mathbf{0} +$

(6) $\sum_{u:\mathbf{Time}} raise \lhd u \rhd RGC'(7, t_t, u, \mathbf{0}, 0)$
$\quad \lhd s = 6 \wedge t_t \leqslant u \leqslant t_t + 5 - d \rhd \delta \lhd \mathbf{0} +$

(7) $ready \lhd (t_g + \frac{90-r}{20}) \rhd RGC'(1, t_t, t_g + \frac{90-r}{20}, d, 90)$
$\quad \lhd s = 7 \rhd \delta \lhd \mathbf{0} +$

(8) $\sum_{t:\mathbf{Time}} app \lhd t \rhd RGC'(8, t, t_g, \mathbf{0}, r)$
$\quad \lhd s = 7 \wedge t_t + \frac{400}{52} \leqslant t \leqslant t_g + \frac{90-r}{20} \rhd \delta \lhd \mathbf{0} +$

(9) $ready \lhd (t_g + \frac{90-r}{20}) \rhd RGC'(2, t_t, t_g + \frac{90-r}{20}, d, 90)$
$\quad \lhd s = 8 \wedge max(t_t, t_g + \frac{90-r}{20}) \leqslant t_t + 5 - d \rhd \delta \lhd \mathbf{0} +$

(10) $\sum_{u:\mathbf{Time}} lower \lhd u \rhd RGC'(3, t_t, u, \mathbf{0}, 20(u - t_g) + r)$
$\quad \lhd s = 8 \wedge t_t \leqslant u \leqslant min(t_t + 5 - d, t_g + \frac{90-r}{20}) \rhd \delta \lhd \mathbf{0} +$

(11) $\sum_{t:\mathbf{Time}} pass \lhd t \rhd RGC'(9, t, t_g, d, r)$
$\quad \lhd s = 3 \wedge t_t + \frac{1000}{52} \leqslant t \leqslant min(t_t + 25, t_g + \frac{r}{20}) \rhd \delta \lhd \mathbf{0} +$

(12) $ready \lhd (t_g + \frac{r}{20}) \rhd RGC'(5, t_t, t_g + \frac{r}{20}, d, 0)$
$\quad \lhd s = 9 \wedge max(t_g + \frac{r}{20}, t_t + \frac{100}{52}) \leqslant t_t + \frac{10}{4} \rhd \delta \lhd \mathbf{0} +$

(13) $\sum_{t:\mathbf{Time}} exit \lhd t \rhd RGC'(10, t, t_g, \mathbf{0}, r)$
$\quad \lhd s = 9 \wedge t_t + \frac{100}{52} \leqslant t \leqslant min(t_t + \frac{10}{4}, t_g + \frac{r}{20}) \rhd \delta \lhd \mathbf{0} +$

$$(14)\quad ready^{\lhd}(t_g + \tfrac{r}{20})\, RGC'(6, t_t, t_g + \tfrac{r}{20}, d, 0) \lhd s = 10 \wedge max(t_t, t_g + \tfrac{r}{20}) \leqslant t_t + 5 - d \rhd \delta^{\lhd}\mathbf{0} +$$

$$(15)\quad \sum_{u:\mathbf{Time}} raise^{\lhd}u\, RGC'(7, t_t, u, \mathbf{0}, 20(t_g - u) + r) \lhd s = 10 \wedge t_t \leqslant u \leqslant min(t_t + 5 - d, t_g + \tfrac{r}{20}) \rhd \delta^{\lhd}\mathbf{0}$$

4.4.3. A detailed invariant

Again, we use an invariant for further reduction of the system equation. Let $I_{RGC}(s, t_t, t_g, d, r)$ be defined by

$$\begin{aligned} d = \mathbf{0} \wedge ((&s = 1 \wedge r = 90) \vee (s = 2 \wedge r = 90) \\ &\vee (s = 3 \wedge t_g \leqslant t_t + 5 \wedge r \leqslant 90) \vee (s = 4 \wedge r = 0) \\ &\vee (s = 5 \wedge r = 0) \vee (s = 6 \wedge r = 0) \\ &\vee (s = 7 \wedge r = 0) \vee (s = 8 \wedge r = 0)). \end{aligned}$$

Note that $I_{RGC}(s, t_t, t_g, d, r)$ implies $\bigvee_{i=1\ldots10} s = \sigma_i$, which was a necessary condition for proving $RGC = RGC'$.

Using the above invariant, we may reduce the equation for RGC' considerably. Let

$$\mathbf{proc}\quad RGC''(s{:}RState, t_t{:}\mathbf{Time}, t_g{:}\mathbf{Time}, r{:}\mathbb{R}) =$$

$$(1')\quad \sum_{t:\mathbf{Time}} app^{\lhd}t\, RGC''(2, t, t_g, 90) \lhd s = 1 \wedge t_t + \tfrac{400}{52} \leqslant t \rhd \delta^{\lhd}\mathbf{0} +$$

$$(2')\quad \sum_{u:\mathbf{Time}} lower^{\lhd}u\, RGC''(3, t_t, u, 90) \lhd s = 2 \wedge t_t \leqslant u \leqslant t_t + 5 \rhd \delta^{\lhd}\mathbf{0} +$$

$$(3')\quad ready^{\lhd}(t_g + \tfrac{r}{20})\, RGC''(4, t_t, t_g + \tfrac{r}{20}, 0) \lhd s = 3 \rhd \delta^{\lhd}\mathbf{0} +$$

$$(4')\quad \sum_{t:\mathbf{Time}} pass^{\lhd}t\, RGC''(5, t, t_g, 0) \lhd s = 4 \wedge t_t + \tfrac{1000}{52} \leqslant t \leqslant t_t + 25 \rhd \delta^{\lhd}\mathbf{0} +$$

$$(5')\quad \sum_{t:\mathbf{Time}} exit^{\lhd}t\, RGC''(6, t, t_g, 0) \lhd s = 5 \wedge t_t + \tfrac{100}{52} \leqslant t \leqslant t_t + \tfrac{10}{4} \rhd \delta^{\lhd}\mathbf{0} +$$

$$(6')\quad \sum_{u:\mathbf{Time}} raise^{\lhd}u\, RGC''(7, t_t, u, 0) \lhd s = 6 \wedge t_t \leqslant u \leqslant t_t + 5 \rhd \delta^{\lhd}\mathbf{0} +$$

$$(7')\quad ready^{\lhd}(t_g + \tfrac{9}{2})\, RGC''(1, t_t, t_g + \tfrac{9}{2}, 90) \lhd s = 7 \rhd \delta^{\lhd}\mathbf{0} +$$

$$(8')\quad \sum_{t:\mathbf{Time}} app^{\lhd}t\, RGC''(8, t, t_g, 0) \lhd s = 7 \wedge t_t + \tfrac{400}{52} \leqslant t \leqslant t_g + \tfrac{9}{2} \rhd \delta^{\lhd}\mathbf{0} +$$

$$(9')\quad ready^{\lhd}(t_g + \tfrac{9}{2})\, RGC''(2, t_t, t_g + \tfrac{9}{2}, 90) \lhd s = 8 \wedge t_g + \tfrac{9}{2} \leqslant t_t + 5 \rhd \delta^{\lhd}\mathbf{0} +$$

$$(10')\quad \sum_{u:\mathbf{Time}} lower^{\lhd}u\, RGC''(3, t_t, u, 20(u - t_g)) \lhd s = 8 \wedge t_t \leqslant u \leqslant min(t_t + 5, t_g + \tfrac{9}{2}) \rhd \delta^{\lhd}\mathbf{0}$$

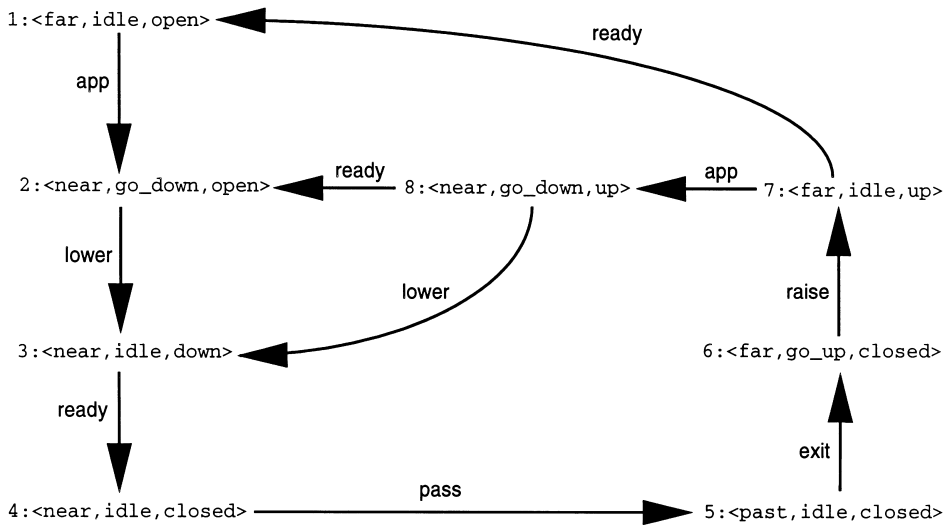

Fig. 5. Schematic transition system of the railroad gate controller.

It holds that

$$I_{TC}(s_t, s_c, t_t, t_c) \wedge I_{RGC}(s, t_t, t_g, d, r) \rightarrow RGC(s, t_t, t_g, d, r) = RGC''(s, t_t, t_g, r).$$

If we abstract from the time conditions we may construct a transition system for the railroad gate controller as in Fig. 5. Each main summand of RGC'' corresponds to a transition. It is easily proved from the specification of RGC'' that all transitions are possible, so that the corresponding terms are not always $\delta \triangleleft \mathbf{0}$.

Consider Fig. 5. We see that after a train has just passed the gates are going up (7). From that state the gates may either reach the highest position (1) or there may come a new train (8). Shortly after the detection of a new train the gates may first completely open and then lower again ($2 \rightarrow 3$). The gates may also lower immediately, so before reaching the highest position.

Some important requirements are obviously satisfied: (a) A train can only pass when the gates are closed ($4 \rightarrow 5$); (b) After a train has left the track and no new train has been detected the gates open and the controller becomes idle again ($7 \rightarrow 1$); (c) As just argued the system adequately reacts when a new train comes shortly after the previous one.

5. Concluding remarks

We were slightly surprised to find that it was possible to describe and analyse hybrid systems in timed μCRL. Using standard process algebraic techniques we could simplify, and hence understand the behaviour of the systems better. Even various correctness and performance issues could be verified.

In our opinion, the case studies in this paper show that timed μCRL may become useful as a formalism for the specification and analysis of hybrid systems. It is unclear to us, however, whether timed μCRL can actually be used to analyse more complex hybrid systems, and to what extent it may provide answers to control theoretic questions.

At this moment, the complexity of the verifications is a little worrying, which is mainly due to the large number of time deadlocks that occur as a result of encapsulation. We saw, for example, in the railroad gate controller, that a simple process such as TC gives rise to a large number of 'main' terms (54 in total, 34 of which consist of time deadlocks). Reduction yields a very acceptable result of only five such terms, but handling the results of the preceding blow-up in the number of terms gives a lot of work. Considering the current 'state of the art' in timed μCRL, the relative simplicity of the ultimate results may not fully justify the complexity of the analysis.

We saw that with each example the number of system components increased with one, and that the complexity of mutual interactions grew significantly with the number of components. In the linearisations of the latter two examples great numbers of conditions on time parameters had to be taken into account.

For a large class of untimed processes a programme already exists for carrying out the linearisation fully automatically. For timed processes the linearisation is considerably more complex, because of the multiple mutual interactions between (the time conditions of) the various summands of the components, but there may be possibilities to extend the current linearisator.

It should be obvious that our major future tasks w.r.t. timed μCRL are to study the problems just mentioned. Hopefully, there are more systematic ways to handle the linearisation of larger multiple-component systems, time conditions and δ-summands.

Throughout this paper we worked without abstraction. It is conceivable that in a setting with abstraction the bottle filling system and the railroad gate controller could be simplified even further. However, despite impressive work in continuous time process algebra, see e.g. [13], the question of how abstraction can be combined with time has not been clarified satisfactorily yet.

Acknowledgements

We thank Paul van de Bosch from Eindhoven University of Technology for providing the bottle filling example. Michel Reniers, from the same university, is thanked for his helpful comments.

Appendix A. Timed μCRL

In this appendix we give a brief summary of timed μCRL as presented in [10], where various basic results are derived. First, the axiom system $p\mathrm{CRL}_t$ for *pico* CRL with time is presented. The following step is to incorporate operators for parallelism

Table 1
Core axioms of $p\mathrm{CRL}_t$

A1	$x + y = y + x$	SUM1	$\sum_{d:D} x = x$
A2	$x + (y + z) = (x + y) + z$	SUM3	$\sum X = \sum X + Xd$
A3	$x + x = x$	SUM4	$\sum_{d:D}(Xd + Yd) = \sum X + \sum Y$
A4	$(x + y) \cdot z = x \cdot z + y \cdot z$	SUM5	$(\sum X) \cdot x = \sum_{d:D}(X\,d \cdot x)$
A5	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	SUM11	$(\forall d \in D\ Xd = Yd) \rightarrow \sum X = \sum Y$
AT6	$x + \delta \lhd \mathbf{0} = x$		
A7	$\delta \cdot x = \delta$	C1	$x \lhd \mathsf{t} \rhd y = x$
		C2	$x \lhd \mathsf{f} \rhd y = y$

and introduce $\mu\mathrm{CRL}_t$. We work in a setting without the silent step τ, and without abstraction or general operators for renaming. We also define a notion of *basic forms* and state that all terms over the signature $\Sigma(p\mathrm{CRL}_t)$ without process variables are provably equal to basic forms.

A.1. Axioms for pCRL with time

Atomic actions are the building blocks of processes. Therefore, axiom systems in process algebra have a set of atomic actions A as a parameter. The actions are parameterised with data, and w.l.o.g. we may assume that all actions have exactly one such parameter. For process variables we use $x, y, z, \ldots$, and for process terms we use $p, q, r, \ldots$. Choice or alternative composition is modelled by $+$, and sequential composition by $\cdot$, which is often omitted from expressions. (We write $\cdot$ only in the tables of axioms.) Deadlock is modelled by δ. Symbols $a, b, c, \ldots$ are used to denote elements from A, or elements from $A \cup \{\delta\}$ (A_δ). We always take care that it is clear to which set they refer.

Basically, Table 1 lists the 'core' axioms of untimed pCRL, with A6 replaced by AT6. Axioms A1–A5 and A7 are well known from process algebra, and axiom AT6 expresses that a deadlock at time $\mathbf{0}$ may always be eliminated from an alternative composition. The $\sum$-operator will be explained below.

Data types in μCRL are algebraically specified in the standard way using sorts, functions and axioms. For data sorts we use $D, E, \ldots$, and for data variables of the respective sorts we use $d, e, \ldots$. Data types are assumed to be non-empty. Two special sorts are assumed in $\mu\mathrm{CRL}_t$: **Bool** and **Time**.

Sort **Bool** contains the constants t ("true") and f ("false"). Typical boolean variables are $\alpha, \beta, \ldots$, and the use of booleans in process expressions may become clear from the axioms C1 and C2 for the conditional construct $\_ \lhd \_ \rhd \_$. For sort **Bool** we assume connectives $\neg, \wedge, \vee, \rightarrow$ with straightforward interpretations, and for the construction of proofs we (implicitly) use the proof theory for μCRL [8], which also provides a rule for structural induction on data terms. For booleans, this implies that we may use the principle of *case distinction* in proofs, i.e., if a formula ϕ holds for both $\alpha = \mathsf{t}$ and

Table 2
Time-related axioms of $p\mathrm{CRL}_t$, where $a \in A_\delta$

Time1	$\mathit{if}\ \ t_1 \leqslant t_2 \wedge t_2 \leqslant t_3 = \mathsf{t}\ \ \mathit{then}\ \ t_1 \leqslant t_3 = \mathsf{t}$
Time2	$\mathbf{0} \leqslant t = \mathsf{t}$
Time3	$t_1 \leqslant t_2 \vee t_2 \leqslant t_1 = \mathsf{t}$
Time4	$\mathit{if}\ \ t_1 \leqslant t_2 \wedge t_2 \leqslant t_1 = \mathsf{t}\ \ \mathit{then}\ \ t_1 = t_2$
Time5	$\mathit{eq}(t_1, t_2) = t_1 \leqslant t_2 \wedge t_2 \leqslant t_1$
Time6	$\mathit{min}(t_1, t_2) = \mathit{if}(t_1 \leqslant t_2, t_1, t_2)$
Time7	$\mathit{if}(\mathsf{t}, t_1, t_2) = t_1$
Time8	$\mathit{if}(\mathsf{f}, t_1, t_2) = t_2$
ATA1	$x = \sum_{t:\mathbf{Time}} x \mathbin{@} t$
ATA2	$a \mathbin{@} t = (a \mathbin{@} t + \delta \mathbin{@} u) \lhd u \leqslant t \rhd a \mathbin{@} t$
ATA3	$a \mathbin{@} t \cdot x = a \mathbin{@} t \cdot (t \gg x)$
ATB1	$a \mathbin{@} t \mathbin{@} u = (a \mathbin{@} t \lhd u \leqslant t \rhd \delta \mathbin{@} t) \lhd t \leqslant u \rhd \delta \mathbin{@} u$
ATB2	$(x + y) \mathbin{@} t = x \mathbin{@} t + y \mathbin{@} t$
ATB3	$(x \cdot y) \mathbin{@} t = x \mathbin{@} t \cdot y$
ATB4	$(\sum_{d:D} Xd) \mathbin{@} t = \sum_{d:D} Xd \mathbin{@} t$
$\gg 1$	$t \gg x = \sum_{u:\mathbf{Time}} x \mathbin{@} u \lhd t \leqslant u \rhd \delta \mathbin{@} t$

f then ϕ holds in general. As a consequence, we have to require that for the data specifications only *minimal* models are considered.

Sort **Time** contains a constant $\mathbf{0}$ ("zero"), which serves as a minimal element for the total ordering $\leqslant$. Axioms for $\leqslant$, *eq* (equality, which we often simply express using "="), *min* (minimum), and *if* (if-then-else) are listed in Table 2. A function $<$ is used to abbreviate terms $t \leqslant u \wedge \neg \mathit{eq}(t,u)$ to $t < u$, and $u \leqslant t \leqslant v$ abbreviates $u \leqslant t \wedge t \leqslant v$. Typical elements of sort **Time** are $t, u, v, \ldots$, and unless stated explicitly, such as in axioms with $\sum_{t:\mathbf{Time}}$, **Time** is treated as a normal μCRL data type.

An expression of the form $p[d_0/d]$ denotes process p with data term d_0 substituted for variable d. *Process-closed terms* are terms without process variables, but possibly with bound and free data variables.

The *at* operator adds time parameters to processes: $p \mathbin{@} t$ should be interpreted as p at time t. Table 2 contains the axioms for the *at* operator. In $p\mathrm{CRL}_t$, we have by axiom ATA1 that $\delta = \sum_{t:\mathbf{Time}} \delta \mathbin{@} t$, so δ models the process that will never do a step, terminate or block. Processes $\delta \mathbin{@} t$ do model deadlocks at time t. Therefore we call them *time deadlocks*.

In general, for $n > 0$ finite sums $p_1 + \ldots + p_n$ are abbreviated by $\sum_{i \in I} p_i$, where $I = \{1, \ldots, n\}$. We define $\sum_{i \in \emptyset} p = \delta \mathbin{@} \mathbf{0}$. In μCRL, a summation construct of the form $\sum_{d:D} p$ is a binder of variable d of data sort D in p. D may be infinite. Finally, the notation $x \subseteq y$ stands for $x + y = y$, so x is a summand of y.

Table 3
Time-related axioms of $\mu\mathrm{CRL}_t$, where $a \in A_\delta$

ATB6	$(x \between y)^\frown t = x^\frown t \between y$
ATB7	$(x \mid y)^\frown t = x^\frown t \mid y$
ATB8	$(x \mid y)^\frown t = x \mid y^\frown t$
ATB9	$\partial_H(x^\frown t) = \partial_H(x)^\frown t$
$\lll 1$	$x \lll a^\frown t = \sum_{u:\mathbf{Time}} x^\frown u \lhd u \leqslant t \rhd x^\frown t$
$\lll 2$	$x \lll (y + z) = x \lll y + x \lll z$
$\lll 3$	$x \lll y \cdot z = x \lll y$
$\lll 4$	$x \lll \sum X = \sum_{d:D} x \lll Xd$

In axioms SUMx distinction is made between *sum operators* $\sum$ and *sum constructs* $\sum_{d:D} p$. The axioms are defined for any sort D. The X in $\sum X$ may be instantiated with functions from some data sort to the sort of processes, such as $\lambda d{:}D.p$, where variable d in p may not become bound by $\sum$. We also have expressions $\sum_{d:D} x$, where some term p that is substituted for x may not contain free variable d. Data terms are considered modulo α-*conversion*, e.g., the terms $\sum_{d:D} p(d)$ and $\sum_{e:D} p(e)$ are equal.

Axiom $\ggg 1$ in Table 2 adds no new identities to the theory, and should only be regarded as a means to simplify certain notations.

We conclude this section with an important identity.

Theorem A.1 (Sum elimination). *It holds that*
$\sum_{d:D} p \lhd d = e \rhd \delta^\frown \mathbf{0} = p[e/d]$.

A.2. Addition of time and operators for parallelism

The axioms of $\mu\mathrm{CRL}_t$ are the axioms of $p\mathrm{CRL}_t$, combined with the axioms in Tables 3 and 4. The signature $\Sigma(\mu\mathrm{CRL}_t)$ is as $\Sigma(p\mathrm{CRL}_t)$, extended with the operators for parallelism and the $\lll$ operator.

For communication we have a binary function γ, which is only defined on *action labels*. In order for a communication to occur between actions $c, c' \in A$, $\gamma(c, c')$ should be defined, and the data parameters of the actions should match according to axiom CF. By definition, the function γ is commutative and associative.

Concurrency is basically described by three operators: the *merge* $\|$, the *left merge* $\between$ and the *communication merge* $|$. The process $p \| q$ symbolises the parallel execution of p and q. It 'starts' with an action of either p or q, or with a communication, or *synchronisation*, between p and q. $p \between q$ is as $p \| q$, but the first action that is performed comes from p.

For the axiomatisation of the left merge $\between$ the auxiliary *before* operator is defined; $p \lll q$ should be interpreted as the process that behaves like p, provided that p can do a step before or at the moment t_0 after which q gets definitively disabled. Otherwise $p \lll q$ becomes a time deadlock at time t_0.

Table 4
Axioms for parallelism of $\mu\mathrm{CRL}_t$, where $a, b \in A_\delta$ and $c, c' \in A$

SUM6	$(\sum X)\mathbin{\lfloor\!\lfloor} x = \sum_{d:D}(Xd\mathbin{\lfloor\!\lfloor} x)$	CF	$c(d) \mid c'(e) = \begin{cases} \gamma(c,c')(d) \lhd eq(d,e) \rhd \delta \\ \quad \text{if sorts of } d \text{ and } e \text{ are equal,} \\ \quad \text{and } \gamma(c,c') \text{ defined} \\ \delta \text{ otherwise} \end{cases}$
SUM7	$(\sum X) \mid x = \sum_{d:D}(Xd \mid x)$		
SUM7′	$x \mid (\sum X) = \sum_{d:D}(x \mid Xd)$		
SUM8	$\partial_H(\sum X) = \sum_{d:D} \partial_H(Xd)$		
CM1	$x \parallel y = x\mathbin{\lfloor\!\lfloor} y + y\mathbin{\lfloor\!\lfloor} x + x \mid y$	CD1	$\delta \mid a = \delta$
CM2	$a \triangleleft t \mathbin{\lfloor\!\lfloor} x = (a \triangleleft t \lessdot x) \cdot x$	CD2	$a \mid \delta = \delta$
CM3	$a \triangleleft t \cdot x \mathbin{\lfloor\!\lfloor} y = (a \triangleleft t \lessdot y) \cdot (t \gtrdot x \parallel y)$		
CM4	$(x + y)\mathbin{\lfloor\!\lfloor} z = x\mathbin{\lfloor\!\lfloor} z + y\mathbin{\lfloor\!\lfloor} z$	DD	$\partial_H(\delta) = \delta$
CM5	$a \cdot x \mid b = (a \mid b) \cdot x$		
CM6	$a \mid b \cdot x = (a \mid b) \cdot x$	D1	$\partial_H(c(d)) = c(d)$ if $c \notin H$
CM7	$a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$	D2	$\partial_H(c(d)) = \delta$ if $c \in H$
CM8	$(x + y) \mid z = x \mid z + y \mid z$	D3	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
CM9	$x \mid (y + z) = x \mid y + x \mid z$	D4	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$

Example A.2. Let $a, b, c \in A$ and t_1, t_2, t_3 be closed terms of sort **Time**. It can be proved that

$$a \triangleleft t_1 \lessdot (b \triangleleft t_2 + c \triangleleft t_3) = a \triangleleft t_1 \lhd t_1 \leqslant max(t_2, t_3) \rhd \delta \triangleleft \mathbf{0} + \delta \triangleleft t_1 \triangleleft max(t_2, t_3).$$

If $t_1 \leqslant max(t_2, t_3)$ then it is easily proved that $a \triangleleft t_1 + \delta \triangleleft t_1 = a \triangleleft t_1$, otherwise the above process equals $\delta \triangleleft max(t_2, t_3)$.

Process $p \mid q$ is as $p \parallel q$, but the first action is a communication between p and q. Encapsulation operators ∂_H block atomic actions in H by renaming them to δ. They are used to enforce communication between processes.

In [10] it is proved that the operators $\parallel$ and $\mid$ are commutative for terms without process variables. In this paper, however, associativity of the operators $\parallel$ and $\mid$ is assumed in the form of axioms. These principles are sound w.r.t. the semantics for $\mu\mathrm{CRL}_t$ as provided in [9]. Also equational proofs of associativity of $\parallel$ and $\mid$ for basic forms must be feasible, but these turn out to be pretty complex.

The various operators of $\Sigma(\mu\mathrm{CRL}_t)$ are listed in order of decreasing binding strength:

$$\triangleleft \quad \cdot \quad \lessdot \quad \{\lhd \rhd, \parallel, \mathbin{\lfloor\!\lfloor}, \mid\} \quad \sum_{d:D} \quad +.$$

Brackets are omitted from expressions according to this convention.

A.3. Basic forms

Here we present some results about the representation of $p\mathrm{CRL}_t$ terms.

Definition A.3. A *basic form* over $\Sigma(p\mathrm{CRL}_t)$ is a process-closed term of the form

$$r = \sum_{i\in I}\sum_{d_1^i:D_1^i}\cdots\sum_{d_{m_i}^i:D_{m_i}^i}\sum_{u:\mathbf{Time}} a_i{\triangleleft}u\, r_i \triangleleft \alpha_i \triangleright \delta{\triangleleft}\mathbf{0} +$$
$$\sum_{j\in J}\sum_{e_1^j:E_1^j}\cdots\sum_{e_{n_j}^j:E_{n_j}^j}\sum_{v:\mathbf{Time}} b_j{\triangleleft}v \triangleleft \beta_j \triangleright \delta{\triangleleft}\mathbf{0}$$

where the $a_i \in A$ and $b_j \in A_\delta$, and the r_i are also basic forms.

In the sequel, we will often write $\overline{\sum}_{d_1,\ldots,d_m} x$ for $\sum_{d_1:D_1}\cdots\sum_{d_m:D_m} x$, and $\overline{d}_m$ for $d_1,\ldots,d_m$. By convention $\overline{\sum}_{\overline{d}_0} x = x$, and it can be proved that the order of the d_k in $\overline{\sum}_{\overline{d}_m} x$ may be changed arbitrarily. So, for example, $\sum_{d_1,d_2} x = \sum_{d_2,d_1} x$. (We take care that no confusion can arise w.r.t. the sorts of the d_k.) For example, if we treat $\sum_{i\in I}$ and $\sum_{j\in J}$ as formal summations we may abbreviate r in the above definition to

$$\overline{\sum}_{i,\overline{d^i}_{m_i},u} a_i{\triangleleft}u\, r_i \triangleleft \alpha_i \triangleright \delta{\triangleleft}\mathbf{0} + \overline{\sum}_{j,\overline{e^j}_{n_j},v} b_j{\triangleleft}v \triangleleft \beta_j \triangleright \delta{\triangleleft}\mathbf{0}.$$

An even more general format for representing basic forms is provided below.

Lemma A.4 (Representation)**.** *Basic form r given in Definition* A.3 *can be represented by*

$$\overline{\sum}_{i,\overline{d}_m,u} a_i{\triangleleft}u\, r_i \triangleleft \alpha_i \triangleright \delta{\triangleleft}\mathbf{0} + \overline{\sum}_{j,\overline{e}_n,v} b_j{\triangleleft}v \triangleleft \beta_j \triangleright \delta{\triangleleft}\mathbf{0},$$

where the sequence $d_1,\ldots,d_m$ contains all data variables from $\bigcup_{i\in I}\{d_1^i,\ldots,d_{m_i}^i\}$, and $e_1,\ldots,e_n$ contains all data variables from $\bigcup_{j\in J}\{e_1^j,\ldots,e_{n_j}^j\}$.

Theorem A.5 (Basic forms)**.** *If q is a process-closed term over $\Sigma(p\mathrm{CRL}_t)$ then there is a basic form p such that $\mu\mathrm{CRL}_t \vdash p = q$.*

A.4. Recursion and RSP

μCRL allows the specification of recursive processes, such as $X(n{:}\mathbb{N},\alpha{:}\mathbf{Bool}) = a(n)X(S(n),\neg\alpha)\,b(\alpha)$, where $a,b \in A$. Recursive processes are usually represented in capitals. The Recursive Specification Principle (RSP) states that every *guarded* specification has a unique solution, i.e., that if two processes satisfy the same system of guarded recursion equations, they must be the same. Consider, for example, process $Y(n{:}\mathbb{N}) = a(n)\,Y(S(n))$. RSP can be used for proving that $X(n) = Y(n)$, so that α actually is a redundant variable, and $b(\alpha)$ can never be performed. For a formal treatment of RSP and more elaborate examples we refer to the literature.

Appendix B. Expansion and encapsulation in timed μCRL

In this appendix we consider timed μCRL processes p and q of the following form:

$$p \stackrel{\text{def}}{=} \overline{\sum}_{i,\overline{d}_i,t} a_i{\triangleleft}t\; p_i \triangleleft \alpha_i \triangleright \delta{\triangleleft}\mathbf{0};$$

$$q \stackrel{\text{def}}{=} \overline{\sum}_{j,\overline{e}_m,u} b_j{\triangleleft}u\, q_j \triangleleft \beta_j \triangleright \delta{\triangleleft}\mathbf{0}.$$

We require that p and q are not equal to $\delta \lhd \mathbf{0}$. If p is of the form $X(e_1,\ldots,e_m)$ and the p_i are all of the form $X(e_1^i,\ldots,e_m^i)$, where the e_k^i are data terms with $sort(e_k^i) = sort(e_k)$ $(k = 1,\ldots,m)$, then we call p a *linear process expression*.

From [10] we have the following Expansion Theorem for $p \| q$.

Theorem B.1 (Expansion). *It holds that*

$$\begin{aligned} p \| q = {} & \sum_{i,j,\overline{d_l},\overline{e_m},u,t} a_i \lhd t \,(t \gg p_i \| q) \lhd t \leqslant u \wedge \alpha_i \wedge \beta_j \rhd \delta \lhd \mathbf{0} + \\ & \sum_{i,j,\overline{d_l},\overline{e_m},t,u} b_j \lhd u \,(u \gg q_j \| p) \lhd u \leqslant t \wedge \alpha_i \wedge \beta_j \rhd \delta \lhd \mathbf{0} + \\ & \sum_{i,j,\overline{d_l},\overline{e_m},t} (a_i \mid b_j) \lhd t \,(p_i \| q_j[t/u]) \lhd \alpha_i \wedge \beta_j[t/u] \rhd \delta \lhd \mathbf{0}. \end{aligned}$$

Encapsulation can be used to enforce synchronisation between two processes. If actions a and b from different system components are meant to synchronously communicate to c, then a and b are put in encapsulation set H, and ∂_H is applied to the system equation. In case the system equation equals $p \| q$ as provided above, we may use the equation below, which allows a more straightforward application of encapsulation.

Let

$$I_H \stackrel{\text{def}}{=} \{i \mid i \in I \;\&\; a_i \in H\} \qquad J_H \stackrel{\text{def}}{=} \{j \mid j \in J \;\&\; b_j \in H\}$$

$$I'_H \stackrel{\text{def}}{=} \{i \mid i \in I \;\&\; a_i \notin H\} \qquad J'_H \stackrel{\text{def}}{=} \{j \mid j \in J \;\&\; b_j \notin H\}$$

$$\Xi \stackrel{\text{def}}{=} \{(i,j) \mid i \in I_H \;\&\; j \in J_H \;\&\; \textit{communication between } a_i \textit{ and } b_j \textit{ is defined}\}$$

For any pair of indices $\xi \in \Xi$ we define ξ^1 and ξ^2 as the first and second projection of ξ. If communication between a_{ξ^1} and b_{ξ^2} is defined, we define $a_{\xi^1} \mid b_{\xi^2} = c_\xi$, where $c_\xi \notin H$.

Theorem B.2 (Encapsulation). *If p and q communicate synchronously then*

$$\partial_H(p \| q) = \sum_{\xi,\overline{d_l},\overline{e_m},t} c_\xi \lhd t \,\partial_H(p_{\xi^1} \| q_{\xi^2}[t/u]) \lhd \alpha_{\xi^1} \wedge \beta_{\xi^2}[t/u] \rhd \delta \lhd \mathbf{0} + \qquad (\Delta 1)$$

$$\sum_{i'_h,j,\overline{d_l},\overline{e_m},u,t} a_{i'_h} \lhd t \,\partial_H(t \gg p_{i'_h} \| q) \lhd t \leqslant u \wedge \alpha_{i'_h} \wedge \beta_j \rhd \delta \lhd \mathbf{0} + \qquad (\Delta 2)$$

$$\sum_{i,j'_h,\overline{d_l},\overline{e_m},t,u} b_{j'_h} \lhd u \,\partial_H(u \gg q_{j'_h} \| p) \lhd u \leqslant t \wedge \alpha_i \wedge \beta_{j'_h} \rhd \delta \lhd \mathbf{0} + \qquad (\Delta 3)$$

$$\sum_{i_h,j,\overline{d_l},\overline{e_m},u,t} \delta \lhd t \lhd t \leqslant u \wedge \alpha_{i_h} \wedge \beta_j \rhd \delta \lhd \mathbf{0} + \qquad (\Delta 4)$$

$$\sum_{i,j_h,\overline{d_l},\overline{e_m},t,u} \delta \lhd u \lhd u \leqslant t \wedge \alpha_i \wedge \beta_{j_h} \rhd \delta \lhd \mathbf{0} \qquad (\Delta 5)$$

We classify the 5 main terms and introduce some additional terminology:

$\Delta 1$: Summands originating from communication between p and q, or *c-summands*;

$\Delta 2$: Summands originating from non-encapsulated actions from p, or *autonomous p-summands*;

$\Delta 3$: Summands originating from non-encapsulated actions from q, or *autonomous q-summands*;

$\Delta 4$: Time deadlocks originating from encapsulated actions from p, or δ_p*-summands*;

$\Delta 5$: Time deadlocks originating from encapsulated actions from q, or δ_q*-summands*.

In general, the δ-summands cannot be removed. A simple example may demonstrate the meaning of these time deadlocks.

Example B.3. Let $H \stackrel{\text{def}}{=} \{a,b\}$, $a \mid b \stackrel{\text{def}}{=} c$, and

$$p \stackrel{\text{def}}{=} \sum_{t:\mathbf{Time}} a \triangleleft t \; p' \triangleleft 1 \leqslant t \leqslant 2 \vee 4 \leqslant t \leqslant 5 \triangleright \delta \triangleleft \mathbf{0};$$

$$q \stackrel{\text{def}}{=} \sum_{u:\mathbf{Time}} b \triangleleft u \; q' \triangleleft u \leqslant 3 \triangleright \delta \triangleleft \mathbf{0}.$$

p can be split into a process p_1 that can do an a-step at $1 \leqslant t \leqslant 2$, and a process p_2 that can do an a-step at $4 \leqslant t \leqslant 5$. So p_1 and q can communicate between times 1 and 2. However, process p_2 cannot do any step before q can do one, and as a consequence of the definition of the $\between$-operator, a time deadlock occurs as soon as q gets definitively disabled: At time 3. Without proof we state that $\partial_H(p \parallel q) = \sum_{t:\mathbf{Time}} c \triangleleft t \, (p' \parallel q'[t/u]) \triangleleft 1 \leqslant t \leqslant 2 \triangleright \delta \triangleleft \mathbf{0} + \delta \triangleleft 3$.

References

[1] R. Alur, D.L. Dill, A theory of timed automata, Theor. Comput. Sci. 126 (1994) 183–235.

[2] R. Alur, T.A. Henzinger, P.-H. Ho, Automatic symbolic verification of embedded systems, IEEE Trans. Software Engng. 22 (1996) 181–201.

[3] J.C.M. Baeten, J.A. Bergstra, Real time process algebra, J. Formal Aspects Comput. Sci. 3 (2) (1991) 142–188.

[4] J.C.M. Baeten, J.A. Bergstra, Discrete time process algebra, Formal Aspects Comput. 8 (2) (1996) 188–208.

[5] M.A. Bezem, J.F. Groote, Invariants in process algebra with data, in: B. Jonsson, J. Parrow (Eds.), Proc. of Concur '94, Lecture Notes in Computer Science, 836, Springer, Berlin, 1994, pp. 401–416.

[6] J.F. Groote, The syntax and semantics of timed μCRL, Technical Report SEN-R9709, CWI, 1997.

[7] J.F. Groote, A. Ponse, The syntax and semantics of μCRL, in: A. Ponse, C. Verhoef, S.F.M. van Vlijmen (Eds.), Algebra of Communicating Processes, Workshops in Computing, Springer, Berlin, 1994, pp. 26–62.

[8] J.F. Groote, A. Ponse, Proof theory for μCRL: a language for processes with data, in: D.J. Andrews, J.F. Groote, C.A. Middelburg (Eds.), Proc. Internat. Workshop on Semantics of Specification Languages, Workshops in Computing, Springer, Berlin, 1994, pp. 232–251.

[9] J.F. Groote, M.A. Reniers, J.J. van Wamel, M.B. van der Zwaag, A theoretical basis for μCRL with time, to appear as Technical Report, CWI, Amsterdam, 2000.

[10] J.F. Groote, J.J. van Wamel, Basic theorems for parallel processes in timed μCRL, Revised version of Technical Report SEN-R9808, CWI, 1999. Available at http://www.cwi.nl/~jfg/publications1.html. Communicated at the WDS'99 workshop in Iaşi, Romania, 1999.
[11] C. Heitmeyer, N.A. Lynch, The generalized railroad crossing – a case study in formal verification of real-time systems, Proc. 15th IEEE Real-Time Systems Symp., San Juan, Puerto Rico, 1994, pp. 120–131.
[12] T.A. Henzinger, P.-H. Ho, H. Wong-Toi, HyTech: a model checker for hybrid systems, Software Tools Technol. Transfer 1 (1997) 110–122.
[13] A.S. Klusener, Models and axioms for a fragment of real time process algebra, Ph.D. Thesis. Eindhoven University of Technology, 1993.
[14] L. Léonard, G. Leduc, An introduction to ET-LOTOS for the description of time-sensitive systems, Comput. Networks ISDN Syst. 29 (1997) 271–292.
[15] X. Nicollin, J. Sifakis, ATP: theory and application, Inform. Comput. 114 (1994) 131–178.
[16] J. Quemada, C. Miguel, D. de Frutos, L. Llana, A Timed LOTOS extension, in: T. Rus, C. Rattray (Eds.), Theories and Experiences for Real-Time System Development, AMAST Series in Computing, 1994, pp. 239–263.
[17] G.M. Reed, A.W. Roscoe, A timed model for communicating sequential processes, Theor. Comput. Sci. 58 (1988) 249–261.